JAPAN SMARTPHONE SECURITY ASSOCIATION

Android Application Secure Design/Secure Coding Guidebook

September 16, 2026 Edition

Japan Smartphone Security Association（JSSEC）

Secure Coding Working Group

Document control number：JSSEC-TECA-SC-GD20260916B

※　本ガイドの内容は執筆時点のものです。サンプルコードを使用する場合はこの点にあらかじめご注意ください。

※　JSSEC ならびに執筆関係者は、このガイド文書に関するいかなる責任も負うものではありません。全ては自己責任にてご活用ください。

※　Android™は、Google, Inc.の商標または登録商標です。また、本文書に登場する会社名、製品名、サービス名は、一般に各社の登録商標または商標です。本文中では®、TM、© マークは明記していません。

※　この文書の内容の一部は、Google, Inc.が作成、提供しているコンテンツをベースに複製したもので、クリエイティブ・コモンズの表示 3.0 ライセンスに記載の条件に従って使用しています。

Contents

1 Introduction **2**
1.1 Building a Secure Smartphone Society . . . 2
1.2 Timely Feedback on a Regular Basis Through the Beta Version . . . 3
1.3 Usage Agreement of the Guidebook . . . 3
About the Articles Corrected/Updated from the August 27, 2025 Edition . . . 4

2 Composition of the Guidebook **6**
2.1 Developer's Context . . . 6
2.2 Sample Code, Rule Book, Advanced Topics . . . 6
2.3 The Scope of the Guidebook . . . 8
2.4 Literature on Android Secure Coding . . . 8
2.5 Importing Sample Code into Android Studio . . . 9

3 Basic Knowledge of Secure Design and Secure Coding **15**
3.1 Android Application Security . . . 15
3.2 Handling Input Data Carefully and Securely . . . 24

4 Using Technology in a Safe Way **26**
4.1 Creating/Using Activities . . . 26
4.2 Receiving/Sending Broadcasts . . . 83
4.3 Creating/Using Content Providers . . . 119
4.4 Creating/Using Services . . . 180
4.5 Using SQLite . . . 229
4.6 Handling Files . . . 245
4.7 Using Intent . . . 287
4.8 Outputting Log to LogCat . . . 299
4.9 Using WebView . . . 308
4.10 Using Notifications . . . 325
4.11 Using Shared Memory . . . 342
4.12 Using NFC and FeliCa . . . 367

5 How to use Security Functions **373**
5.1 Creating Authentication (Password/Passkey) Screens . . . 373
5.2 Permission and Protection Level . . . 380
5.3 Add In-house Accounts to Account Manager . . . 416
5.4 Communicating via HTTPS . . . 435
5.5 Handling privacy data . . . 469
5.6 Using Cryptography . . . 519
5.7 Using biometric authentication features . . . 552

6 Difficult Problems **569**

6.1 Risk of Information Leakage from Clipboard . . . 569

Revision history **580**
Published by . . . 586
Authors of the August 27, 2025 Edition . . . 587
Authors of January 29 2025 Edition . . . 588
Authors of February 29 2024 Edition . . . 589
Authors of August 29 2022 Edition . . . 590
Authors of January 17 2022 Edition . . . 591
Authors of October 19 2021 Edition . . . 592
Authors of November 1, 2020 Edition . . . 593
Authors of September 1 2019 Edition . . . 594
Authors of September 1, 2018 Edition . . . 595
Authors of February 1, 2018 Edition . . . 596
Authors of February 1, 2017 Edition . . . 597
Authors of September 1, 2016 Edition . . . 598
Authors of February 1, 2016 Edition . . . 599
Authors of June 1, 2015 Edition . . . 600
Authors of July 1, 2014 English Edition . . . 601
Authors of April 1, 2014 English Edition . . . 602
Authors of April 1, 2013 Japanese Edition . . . 603
Authors of November 1, 2012 Japanese Edition . . . 604
Authors of June 1, 2012 Japanese Edition . . . 605

September 16, 2026 Edition
Japan Smartphone Security Association (JSSEC)
Secure Coding Working Group

- The content of this guide is up to date as of the time of publication, but standards and environments are constantly evolving. When using sample code, make sure you are adhering to the latest coding standards and best practices.
- JSSEC and the writers of this guide are not responsible for how you use this document. Full responsibility lies with you, the user of the information provided.
- Android is a trademark or a registered trademark of Google Inc. The company names, product names and service names appearing in this document are generally the registered trademarks or trademarks of their respective companies. Further, the registered trademark ®, trademark (TM) and copyright © symbols are not used throughout this document.
- Parts of this document are copied from or based on content created and provided by Google, Inc. They are used here in accordance with the provisions of the Creative Commons Attribution 3.0 License

1 Introduction

1.1 Building a Secure Smartphone Society

This guidebook is a collection of tips concerning the know-how of secure designs and secure coding for Android application developers. Our intent is to have as many Android application developers as possible take advantage of this, and for that reason we are making it public.

In recent years, the smartphone market has witnessed a rapid expansion, and its momentum seems unstoppable. Its accelerated growth is brought on due to the diverse range of applications. An unspecified large number of key functions of mobile phones that were once not accessible due to security restrictions on conventional mobile phones have been made open to smartphone applications. Subsequently, the availability of varied applications that were once closed to conventional mobile phones is what makes smartphones more attractive.

With great power that comes from smartphone applications comes great responsibility from their developers. The default security restrictions on conventional mobile phones had made it possible to maintain a relative level of security even for applications that were developed without security awareness. As it has been aforementioned with regard to smartphones, since the key advantage of a smartphone is that they are open to application developers, if the developers design or code their applications without the knowledge of security issues then this could lead to risks of users' personal information leakage or exploitation by malware causing financial damage such as from illicit calls to premium-rate numbers.

Compared to the iPhone, Android smartphones require application developers to pay more attention to security. This is because Android smartphones have more mobile phone features open to application developers compared to the iPhone, and applications can be released on Google Play (formerly Android Market) without any screening compared to the App Store, leaving application security almost entirely in the hands of the application developers.

In conjunction with the rapid growth of the smartphone market, there has been a sudden influx of software engineers from different areas in the smartphone application development market. As a result, there is an urgent call for the sharing knowledge of secure design and consolidation of secure coding know-how for specific security issues related to mobile applications.

Due to these circumstances, Japan's Smartphone Security Association (JSSEC) has launched the Secure Coding Group, and by collecting the know-how of secure design as well as secure coding of Android applications, it has decided to make all of the information public with this guidebook. It is our intention to raise the security level of many of the Android applications that are released in the market by having many Android application developers become acquainted with the know-how of secure design and coding. As a result, we believe we will be contributing to the creation of a more reliable and safe smartphone society.

1.2 Timely Feedback on a Regular Basis Through the Beta Version

We, the JSSEC Secure Coding Group, will do our best to keep the content contained in the Guidebook as accurate as possible, but we cannot make any guarantees. We believe it is our priority to publicize and share the know-how in a timely fashion. Equally, we will upload and publicize what we consider to be the latest and most accurate correct information at that particular juncture, and will update it with more accurate information once we receive any feedback or corrections. In other words, we are taking the beta version approach on a regular basis. We think this approach would be meaningful for many of the Android application developers who are planning on using the Guidebook.

The latest version of the Guidebook and sample codes can be obtained from the URL below.

- https://www.jssec.org/dl/android_securecoding_en.pdf Guidebook (English)
- https://www.jssec.org/dl/android_securecoding_en.zip Sample Codes (English)

The latest Japanese version can be obtained from the URL below.

- https://www.jssec.org/dl/android_securecoding.pdf Guidebook (Japanese)
- https://www.jssec.org/dl/android_securecoding.zip Sample Codes (Japanese)

1.3 Usage Agreement of the Guidebook

The user must agree to the following two terms and conditions when using this Guidebook.

1. This Guidebook may contain inaccuracies. Please use this information at your own risk.
2. If you find any errors contained in this Guidebook, please contact us by e-mail using the contact information below. Please note, however, that we cannot promise to respond to you or to make any corrections.

Japan Smartphone Security Association (JSSEC)

Contact Information

URLhttps://www.jssec.org/contact

About the Articles Corrected/Updated from the August 27, 2025 Edition

This section contains the revisions that were found by checking the facts against the previous version of the article. Each revised article incorporates the results of ongoing research by the authors as well as a wide range of valuable suggestions from readers. In particular, the suggestions that we received are the most important factors in making this revised edition a more practical-oriented guide with a higher degree of completeness.

Readers who have been developing apps based on the previous version are requested to take a particular look at the list of revised articles below. The items listed here do not include corrections for typographical errors, changes in organization, or simple improvements in wording.

Any comments, opinions, or suggestions on this Guidebook are greatly appreciated.

List of Corrected/Updated Articles

Table 1.3.1: List of Corrected/Updated Articles

Section to be Corrected in the August 27, 2025 Edition	Corrected/Updated Article in This Revised Edition	Summary of Correction/Update
(N/A)	4.1.3.7. *Strengthening Restrictions on Background Activity Launch (BAL)*	Added descriptions regarding the tightening of background activity launch (BAL) restrictions.
(N/A)	4.2.3.10. *Autonomous Re-pairing for Bluetooth Bond Loss*	Added descriptions regarding autonomous re-pairing for Bluetooth bond loss.
(N/A)	4.3.3.2. *Tightening of Contacts Provider (CP2)*	Added descriptions regarding the tightening of Contact Provider (CP2) access.
(N/A)	4.4.3.5. *Strengthening Background Audio Restrictions*	Added descriptions regarding the tightening of background audio restrictions.
(N/A)	4.7.3.8. *Restrictions on Implicit URI Grants*	Added descriptions regarding the restriction of implicit URI permission grants.
(N/A)	4.7.3.9. *Cross-Profile Loopback Restriction*	Added descriptions regarding cross-profile loopback restrictions.
(N/A)	4.12. *Using NFC and FeliCa*	Added descriptions regarding using NFC and FeliCa.
(N/A)	5.2.3.9. *SMS OTP (One-Time Password) Protection*	Added descriptions regarding the enhancement of SMS OTP (One-Time Password) protection.
(N/A)	5.4.3.9. *Enabling ECH (Encrypted Client Hello)*	Added descriptions regarding enabling ECH (Encrypted Client Hello).
(N/A)	5.4.3.10. *Certificate Transparency (CT) Compliance*	Added descriptions regarding the automatic verification of Certificate Transparency (CT).
(N/A)	5.4.3.11. *Deprecation of usesCleartextTraffic and Permitting HTTP Communication*	Added descriptions regarding the deprecation of usesCleartextTraffic.
(N/A)	5.5.3.15. *Transition to Contact Picker and Proper Handling of Contact Data*	Added descriptions regarding the migration to Contact Picker and proper handling of contact data.
(N/A)	5.6.3.10. *Per-App Android Keystore (Key) Limits*	Added descriptions regarding per-app keystore (key) limits.
• 4.6.3.7. Enhancing Safety of DCL (Dynamic Code Loading)	4.6.3.7. *Enhanced Safety of DCL (Dynamic Code Loading)*	Reviewed the content regarding making native DCL (Dynamic Code Loading) safer.
• 5.1. Creating a Password Input Screen	5.1. *Creating Authentication (Password/Passkey) Screens*	Added descriptions regarding the setting to hide passwords on physical devices.
• 5.4.1.1. Communicating via HTTP	5.4.1.1. *Communicating via HTTP*	Reviewed the content regarding the handling of usesCleartextTraffic deprecation in HTTP communication.
• 5.4.1.2. Communicating via HTTPS	5.4.1.2. *Communicating via HTTPS*	Reviewed the content regarding ECH and automatic verification of Certificate Transparency (CT) in HTTPS communication.
• 5.5.3.6. Location Access	5.5.3.6. *Location Access*	Reviewed the content regarding the introduction of the Location Button in location access.
• 5.5.3.7. Local Network Access Permission	5.5.3.7. *Local Network Access Permission*	Reviewed the content regarding the requirement of local network permission.
• 4.1.3.7. Safely Launching Activities from Background	(N/A)	Deleted the article regarding safely launching activities from the background.

2 Composition of the Guidebook

2.1 Developer's Context

Many guidebooks that have been written on secure coding include warnings about harmful coding practices and their suggested revisions. Although this approach can be useful at the time of reviewing the source code that has already been coded, it can be confusing for developers that are about to start coding, as they do not know which article to refer to.

The Guidebook has focused on the developer's context of "What is a developer trying to do at this moment?" Equally, we have taken steps to prepare articles that are aligned with the developer's context. For example, we have divided articles into project units by presuming that a developer will be involved in operations such as "Creating/Using Activities", "Using SQLite", etc.

We believe that by publishing articles that support the developer's context, developers will be able to easily locate necessary articles that will be instantly useful in their projects.

2.2 Sample Code, Rule Book, Advanced Topics

Each article is made up of three sections: Sample Code, Rulebook, and Advanced. If you are in a hurry, please refer to the Sample Code and Rulebook. The content is broken down into reusable patterns to a certain extent. If you have issues that do not fit into the Sample Code and Rulebook sections, please refer to the Advanced section. It contains materials to consider when considering how to solve individual issues.

Unless otherwise noted, the sample code and article content are intended for Android 7.0 (API Level 24) and later. Please note that operation has not been confirmed for versions earlier than Android 7.0 (API Level 24), and the countermeasures may not be effective. Also, even if the version is within the target range, please verify operation on the device where it is installed and use it at your own risk.

Additionally, starting with Android 14 and Android 15, there are minimum targetSdkVersion requirements for apps that can be installed, which aims to prevent malware from targeting older API levels to circumvent newer security protections and eliminate such malware across the board.

The targetSdkVersion that can be installed for each Android version is as follows:

Android Version	Installable targetSdkVersion
Android 7.0 - 13	No restrictions
Android 14	targetSdkVersion 23 or later
Android 15	targetSdkVersion 24 or later

2.2.1 Sample Code

Sample code that serves as the basic model within the developer's context and functions as the theme of an article is published in the Sample Code section. If there are multiple patterns, we have provided source code for the different patterns and classified them accordingly. We have strived to make our commentaries as simple as possible. For example, when we want to direct the reader's attention to a security issue that requires attention, a bullet-point number will appear next to "**Point**" in the article. We will also comment on the sample code that corresponds to the bullet-point number by writing "***** Point (Number) *****." Please note that a single point may correspond to multiple pieces of sample code. There are sections throughout the entire source code, albeit very little compared to the entire code, which requires our attention for security. In order to be able to survey the sections that call for scrutiny, we try to post the entire class unit of sample code.

Please note that only a portion of sample code is posted in the Guidebook. A compressed file, which contains the entire sample code, is made public in the URL listed below. It is made public by the Apache License, Version 2.0; therefore, please feel free to copy and paste it. Please note that we have minimized the code for error processing in the sample code to prevent it from becoming too long.

- https://www.jssec.org/dl/android_securecoding_en.zip Sample Codes Archive

The projects/keystore file that is attached in the sample code is the keystore file that contains the developer key for the signature of the APK. The password is "android." Please use it when singing the APK in the In-house sample code.

We have provided the keystore file, debug.keystore, for debugging purposes. When using Android Studio for development, it is convenient for verifying the operational capability of the In-house sample code if the keystore is set for each project. In addition, for sample code that is comprised of multiple APKs, it is necessary to match the android:debuggable setting contained inside each AndroidManifest.xml in order to verify the cooperation between each APK. If the android:debuggable setting is not explicit set when installing the APK from Android Studio, it will automatically become android:debuggable= "true."

For embedding the sample code as well as keystore file into Android Studio, please refer to "2.5. *Importing Sample Code into Android Studio*".

2.2.2 Rule Book

Rules and matters that need to be considered regarding security within the developer's context will be published in the Rule Book section. Rules to be handled in that section will be listed in a table format at the beginning and will be divided into two levels: "Required" and "Recommended." The rules will consist of two types of affirmative and negative statements. For example, an affirmative statement that expresses that a rule is required will say "Required." An affirmative statement that expresses a recommendation will say "Recommended." For a negative statement that expresses the requisite nature of the rule would say, "Definitely not do." For a negative sentence that expresses a recommendation would say, "Not recommended." Since these differentiations of levels are based on the subjective viewpoint of the author, it should only be used as a point of reference.

Sample code that is posted in the Sample Code section reflect these rules and matters that need to be considered, and a detailed explanation on them is available in the Rule Book section. Furthermore, rules and matters that need to be considered that are not dealt with in the Sample Code section are handled in the Rule Book section.

2.2.3 Advanced Topics

Items that require our attention, but that could not be covered in the Sample Code and Rule Book sections within the developer's context will be published in the Advanced Topics section. The Advanced Topics section can be utilized to explore ways to solve separate issues that could not be solved in the Sample Code or Rule Book sections. For example, subject matters that contain personal opinions as well as topics on the limitations of Android OS in relation the developer's context will be covered in the Advanced Topics section.

Developers are always busy. Many developers are expected to have basic knowledge of security and produce many Android applications as quickly as possible in a somewhat safe manner rather than to really understand the deep security matters. However, there are certain applications out there that require a high level of security design and implementation from the beginning. For developers of such applications, it is necessary for them to have a deep understanding concerning the security of Android OS.

In order to benefit both developers who emphasize development speed and also those who emphasize security, all articles of the Guidebook are divided into the three sections of Sample Code, Rule Book, and Advanced Topics. The aim of the Sample Code and Rule Book sections is to provide generalizations about security that anyone can benefit from and source code that will work with a minimal amount of customization and hopefully by just copying and pasting. In the Advanced Topics section, we offer materials that will help developers think in a certain way when they are facing specific problems. It is the aim of the Advanced Topics section to help developers examine optimal secure design and coding when they are involved in building individual applications.

2.3 The Scope of the Guidebook

The purpose of the Guidebook is to collect security best practices that are necessary for general Android application developers. Consequently, our scope is focused mainly on security tips (The "Application Security" section in figure below) for the development of Android applications that are distributed primarily in a public market.

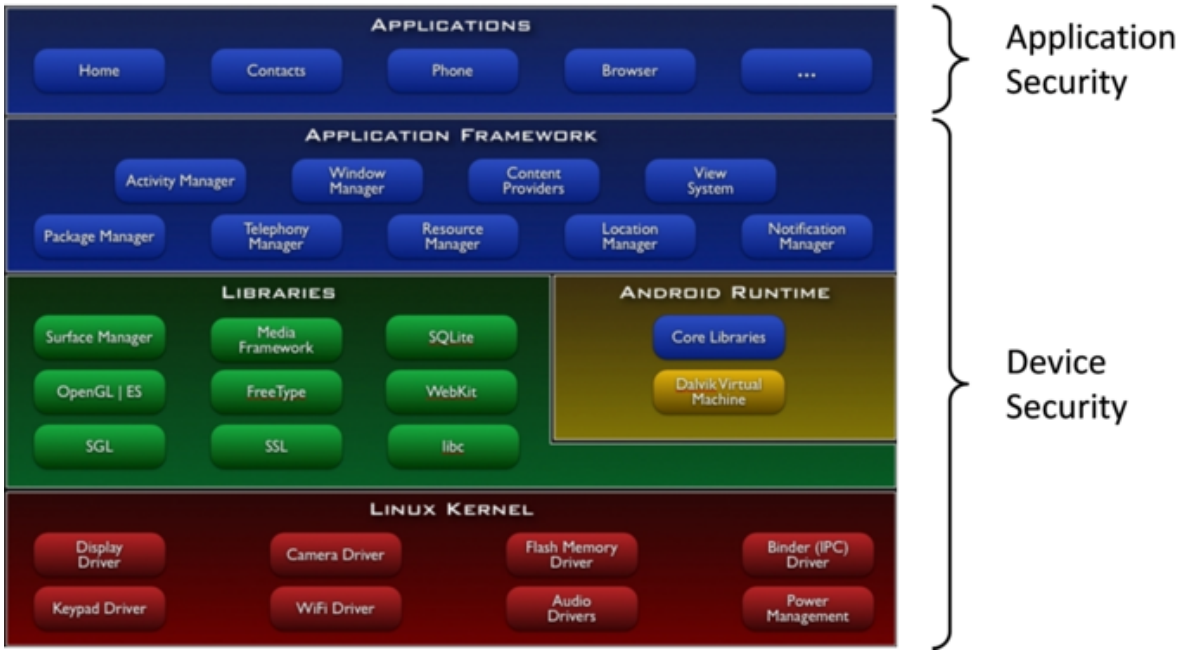


Fig. 2.3.1: Main Components of the Android Platform

Security regarding the implementation of components in the "Device Security" of the above figure is outside the scope of this guidebook. There are differences in the viewpoint of security between general applications that are installed by users and pre-installed applications by device manufacturers. The Guidebook only handles the former and does not deal with the latter. In the current version, tips only on the implementation by Java are posted, but in future versions, we plan on posting tips on JNI implementations as well.

Also as of now we do not handle threats that results from an attacker obtaining root privileges. We will assume the premise of a secure Android device in which it is not possible to obtain root privileges and base our security advice on utilizing the Android OS security model. For handling of assets and threats, we have provided a detailed description on "3.1.3. *Asset Classification and Protective Countermeasures*."

2.4 Literature on Android Secure Coding

Since we are not able to discuss all of Android's secure coding in the Guidebook, we recommend that you read the literature mentioned below in conjunction with the Guidebook.

Android Security: Anzenna Application Wo Sakusei Surutameni (Secured Programming in Android)

Author: Tao Software Co., Ltd. ISBN: 978-4-8443-3134-6
http://www.amazon.co.jp/dp/4844331345/

The CERT Oracle Secure Coding Standard for Java
Authors: Fred Long, Dhruv Mohindra, Robert C. Seacord, Dean F.Sutherland, David Svoboda
http://www.amazon.com/dp/0321803957

OWASP Mobile Application Security Verification Standard (MASVS)
Author: The OWASP Foundation
https://github.com/OWASP/owasp-masvs/

OWASP Top 10
Author: The OWASP Foundation
https://owasp.org/Top10/

10 Major Checkpoints for Mobile Application Development 2023
Author: JSSEC Technical Subcommittee, 10 Major Checkpoints for Mobile Application Development 2023 Selection Committee
https://www.jssec.org/mobile-apps-10checkpoint2023

2.5 Importing Sample Code into Android Studio

This section explains the procedure for importing sample code into Android Studio. The sample code is divided into multiple projects according to their purpose. The method for importing these projects is described in 2.5.1. *Importing Sample Projects*.

2.5.1 Importing Sample Projects

2.5.1.1 Download the Sample Code

Obtain the sample code from the URL introduced in 2.2.1. *Sample Code*.

2.5.1.2 Extract the Sample Code

Right-click the sample code compressed in a Zip file, then click “Extract All” from the displayed menu.

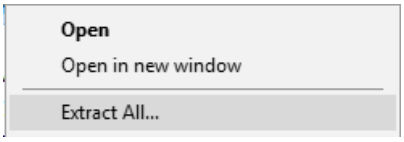


Fig. 2.5.1: Extracting the Sample Code

2.5.1.3 Specify the Extraction Destination

Here, a workspace named “C:android_securecoding” will be created. Specify “C:” and click the “Extract” button.

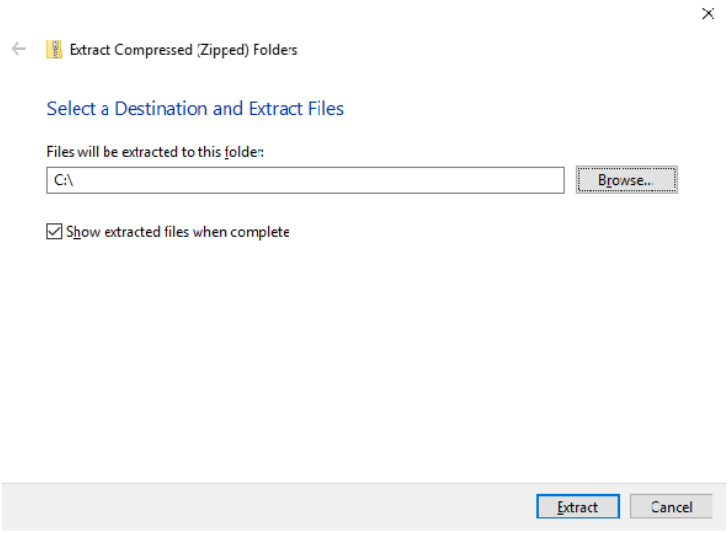


Fig. 2.5.2: Specifying the Extraction Destination

When you click the “Extract” button, a folder named “android_securecoding” will be created directly under “C:”.

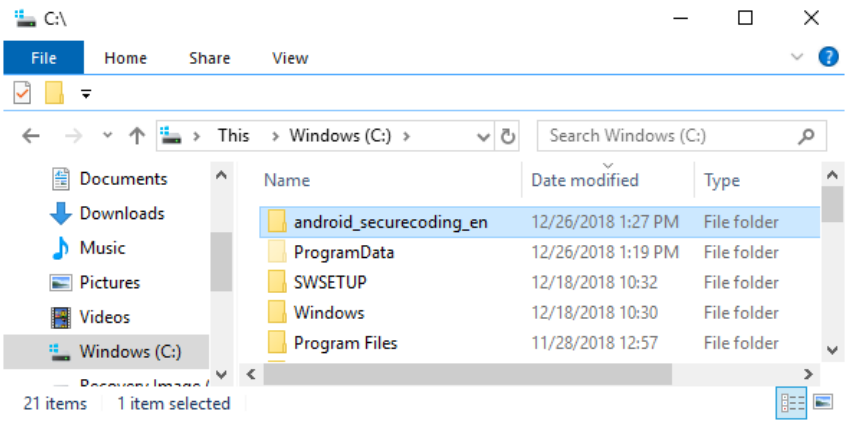


Fig. 2.5.3: android_securecoding Folder

The “android_securecoding” folder contains the sample code.

For example, if you want to refer to the sample code for 4.1.1.4. *Creating/Using In-house Activities* in 4.1. *Creating/Using Activities* please look here:

```
android_securecoding
    └ Create Use Activity
        └ Activity InhouseActivity
```

As shown above, the “android_securecoding” folder contains subfolders for each chapter where sample code projects are arranged accordingly.

2.5.1.4 Build the Sample Code

Start Android Studio from the Start menu or desktop icon.

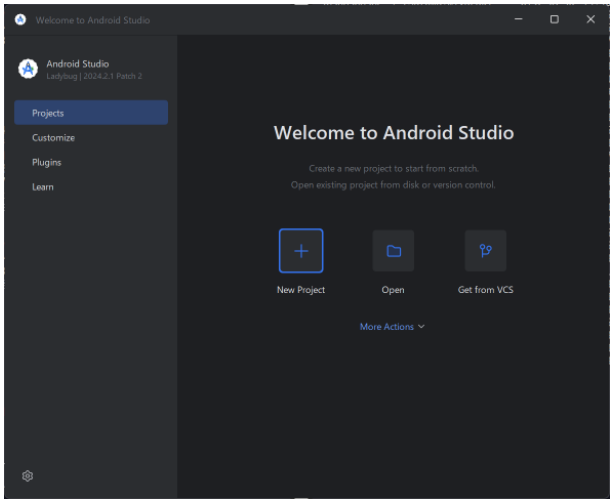


Fig. 2.5.4: Starting Android Studio

Click “Open.”

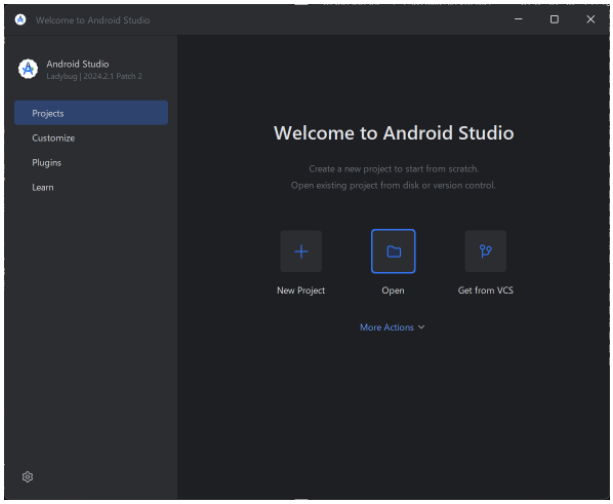


Fig. 2.5.5: Clicking Open

Select the project folder to open.

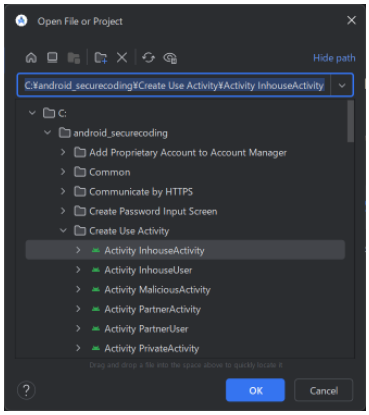


Fig. 2.5.6: Selecting the Project Folder

The project will open.

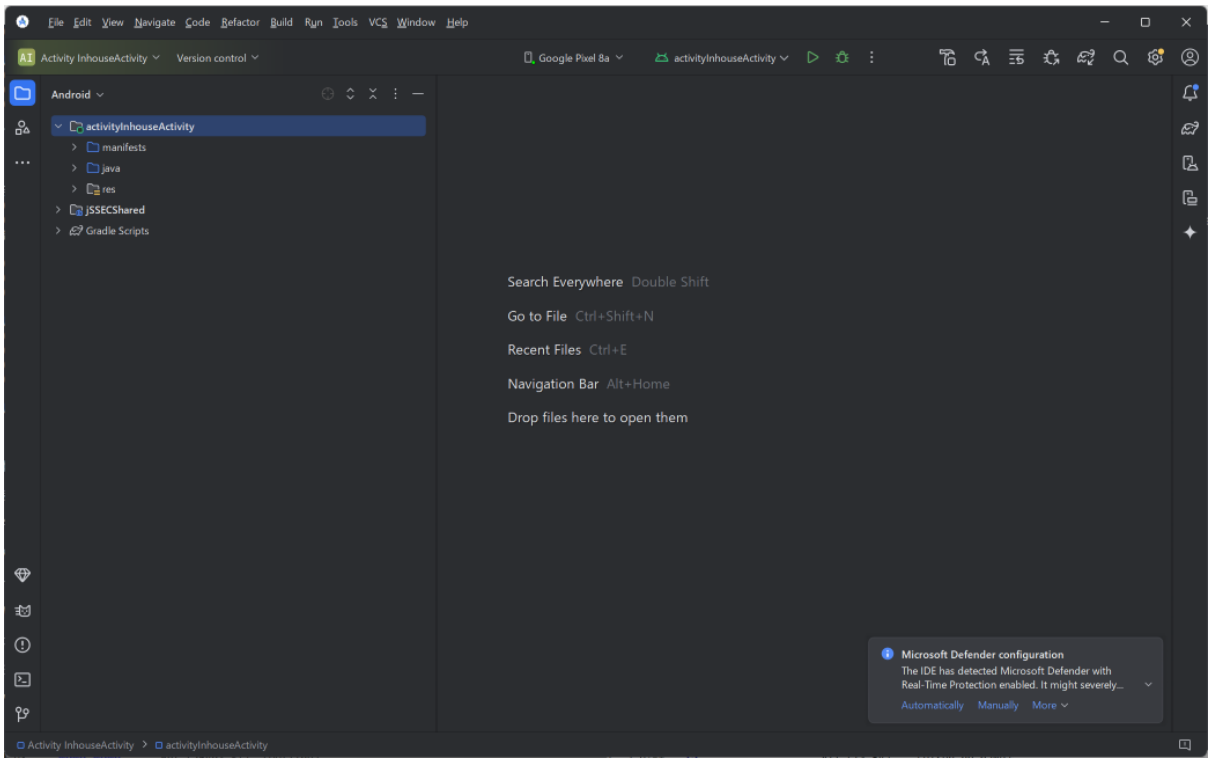


Fig. 2.5.7: Project Opened

To run the app created from the sample code on an Android device or emulator, signing is required. Set the debug key file “debug.keystore” used for signing in the Android Studio project.

Click File -> Project Structure…

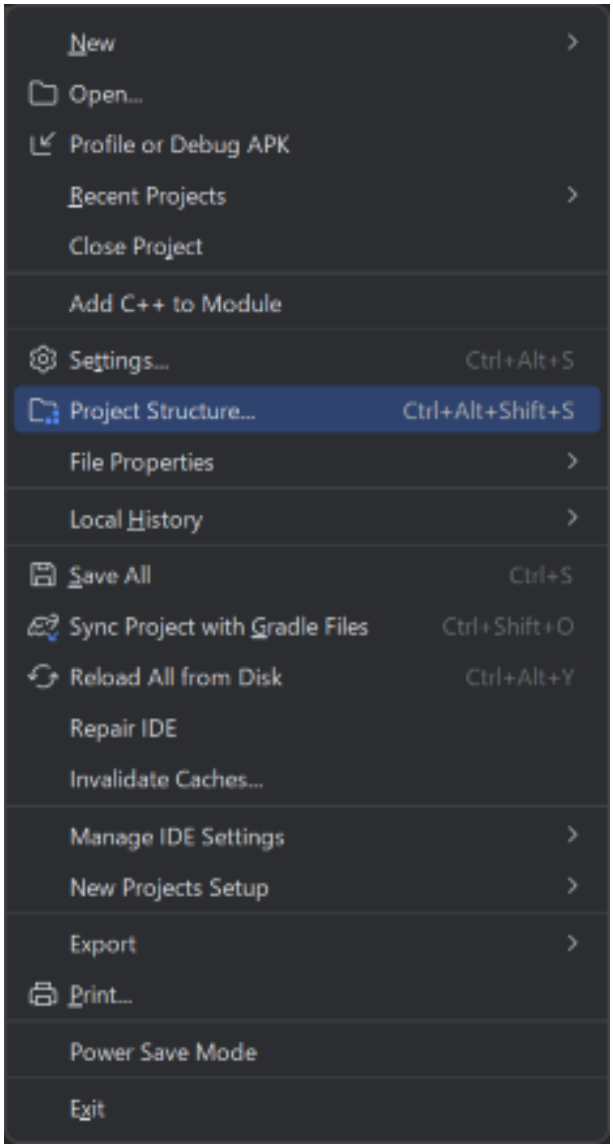


Fig. 2.5.8: File -> Project Structure

Select “debug.keystore” as the Store File and click OK.

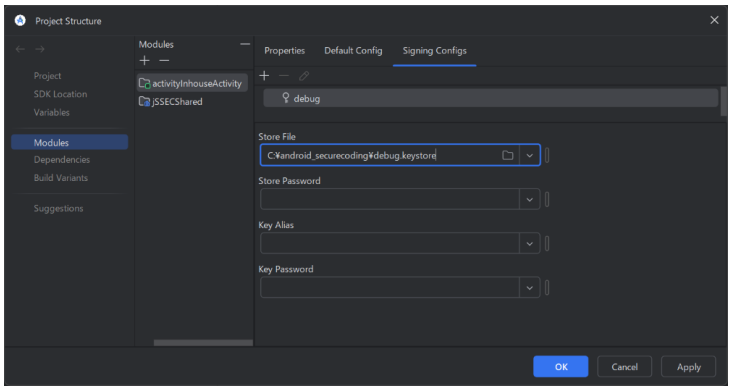


Fig. 2.5.9: Selecting debug.keystore

Click Build -> Make Project…

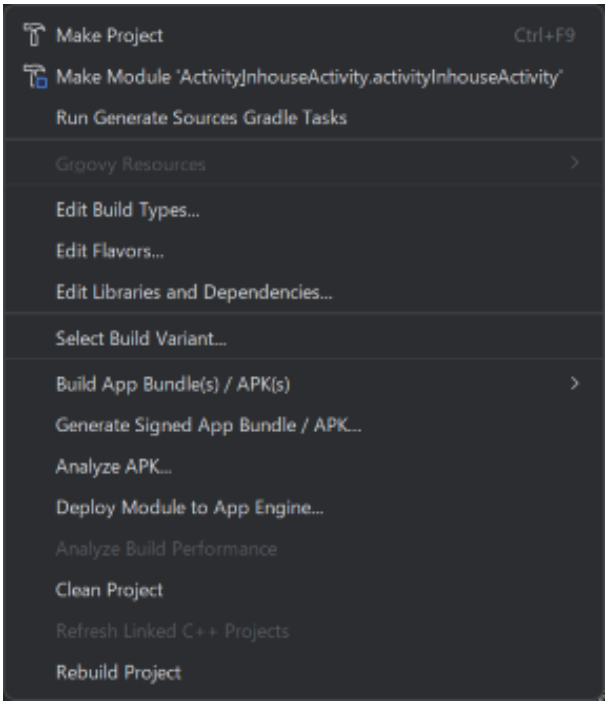


Fig. 2.5.10: Clicking Make Project

3 Basic Knowledge of Secure Design and Secure Coding

Although the Guidebook is a collection of security advice concerning Android application development, this chapter will deal with the basic knowledge on general secure design and secure coding of Android smartphones and tablets. Since we will be referring to secure design and coding concepts in the later chapters we recommend that you familiarize yourself with the content contained in this chapter first.

3.1 Android Application Security

There is a commonly accepted way of thinking when examining security issues concerning systems or applications. First, we need to have a grasp over the objects we want to protect. We will call these "assets". Next, we want to gain an understanding over the possible attacks that can take place on an asset. We will call these "threats". Finally, we will examine and implement measures to protect "assets" from the various "threats". We will call these "countermeasures".

What we mean by "countermeasures" here is secure design and secure coding, and will deal with these subjects after Chapter 4. In this section, we will focus on explaining "assets" and "threats".

3.1.1 "Asset": Object of Protection

There are two types of "objects of protection" within a system or an application: "information" and "functions". We will call these "information assets" and "function assets". "An information asset" refers to the type of information that can be referred to or changed only by people who have permission. It is a type of information that cannot be referred to or changed by anyone who does not have the permission. "A function asset" refers to a function that can be used only by people who have permission and no one else.

Below, we will introduce types of information assets and functional assets that exist in Android smartphones and tablets. We would like you to use the following as a point of reference to deliberate on matters with regard to assets when developing a system that utilizes Android applications or Android smartphones/tablets. For the sake of simplicity, we will collectively call Android smartphones/tablets as Android smartphones.

3.1.1.1 Information Assets of the Android Smartphone

Table 3.1.1 and Table 3.1.2 are examples of information contained in Android smartphones. This information is personal information, privacy information, or similar information about smartphone users and must be properly protected.

Table 3.1.1: Examples of Information Managed by Android Smartphones

Information	Remarks
Phone number	Telephone number of the smartphone itself
Call history	Time and date of incoming and outgoing calls as well as phone numbers
IMEI	Device ID of the smartphone
IMSI	Subscriber ID
Sensor information	GPS, geomagnetic, rate of acceleration, etc.
Various setup information	Wi-Fi setting value, etc...
Account information	arious account information, authentication information, etc.
Media data	Pictures, videos, music, recording, etc…
…	

Table 3.1.2: Examples of Information Managed by Applications

Information	Remarks
Contacts	Contacts of acquaintances
E-mail address	User's e-mail address
Web bookmarks	Bookmarks
Web browsing history	Browsing history
Calendar	Plans, to-do list, events, etc.
Facebook	SNS content, etc.
X	SNS content, etc.
…	

The information in Table 3.1.1 is mainly contained in the Android smartphone itself or on the SD card, while the information in Table 3.1.2 is mainly managed by applications. Especially for the information in Table 3.1.2, the more applications that are installed, the more information will be stored in the device.

Table 3.1.3 shows the information contained in a single Contacts entry. This information is not about the smartphone user, but about the smartphone user's acquaintances, friends, and others. In other words, the smartphone contains information not only about the user, but also about other people, and so caution is necessary.

Table 3.1.3: Example of Information Contained in a Single Contacts Entry

Information	Description
Telephone numbers	Home, mobile, work, fax, MMS,…
Email addresses	Home, work, mobile,…
Profile images	Thumbnail image, large image,…
Instant messengers	Messages, WhatsApp, LINE, Facebook Messenger, WeChat, Telegram,…
Nickname	Abbreviations, initials, maiden name, alias,…
Addresses	Country, postal code, region, province, town, street,…
Groups	Favorites, family, friends, colleagues,…
Websites	Blog, profile site, home page, FTP server, home, office,…
Events	Birthdays, anniversaries, other,…
People involved with	Spouse, children, father, mother, manager, assistant, live-in relationship, partner,…
SIP addresses	Home, work, other,…
…	…

In the previous explanations, we have mainly presented information about smartphone users, but applications also handle information other than that of users.

Fig. 3.1.1 shows the information managed by an application, which can be roughly divided into a program part and a data part. The program part is mainly information about the application manufacturer, and the data part is mainly information about the user. Since some of the application manufacturer's information may not be used by the user without permission, such information must be protected so that the user cannot refer to or change it.

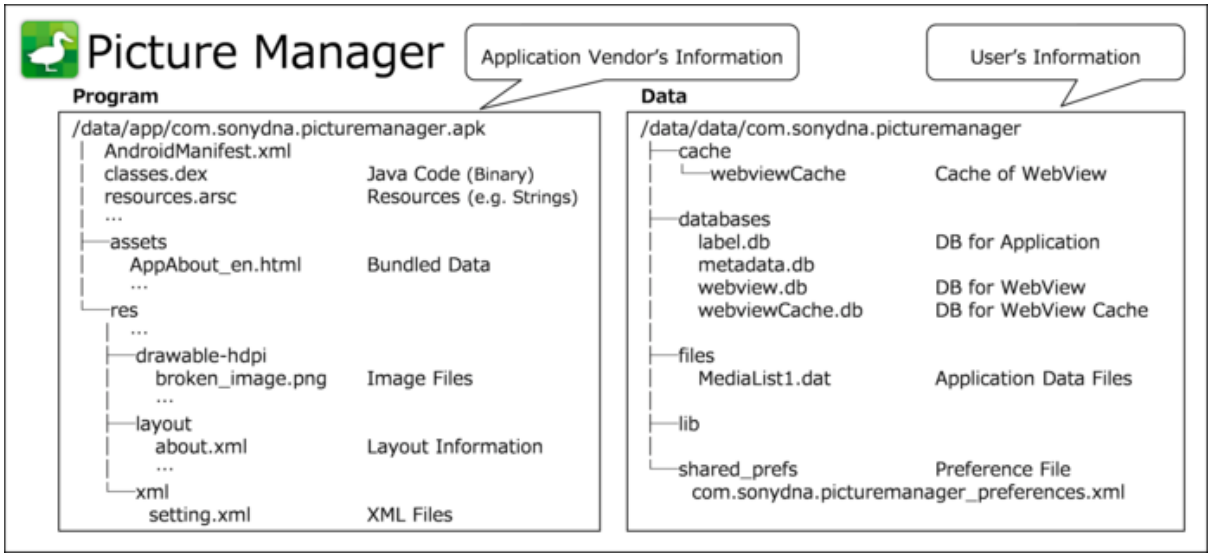


Fig. 3.1.1: Information Contained in an Application

When creating an Android application, it is important to note that not only the information managed by the application itself, shown in Fig. 3.1.1, but also the information obtained from the Android smartphone itself and other applications, shown in Table 3.1.1, Table 3.1.2, and Table 3.1.3.

3.1.1.2 Function Assets of an Android Smartphone

Table 3.1.4 is an example of functions provided by Android OS to the application. If these functions are exploited by malware or other malicious software, damages including unintended charges and compromise of privacy may occur. For this reason, function assets must be protected appropriately as with information assets.

Table 3.1.4: Example of Functions Provided by Android OS to Applications

Function	Description
Send/receive SMS messages	Sends and receives SMS messages
Camera	Shoots photos and videos
Phone call	Makes a phone call
Volume change	Adjusts the device volume
Network communication	Connects to the Internet or local networks
Read phone number/device status	Acquires the phone number and device status
Location information acquisition (GPS, etc.)	Acquires the current location
SD card writing	Writes data to external storage
Bluetooth communication	Communicates with Bluetooth devices
System setting change	Changes various device settings
NFC/FeliCa communication	Uses near-field communication
Log data reading	Reads system or application logs
Internet communication (SIP)	Makes a call using SIP
Running application information acquisition	Acquires information of currently running applications
Others	Various other functions exist besides the above

In addition to functions provided to the application from Android OS, inter-application communication functions of the Android application are included as function assets. The Android application enables use of functions available within the application from other applications. This mechanism is called inter-application communication. While this function is useful, there are cases in which functions that are to be used solely within the application can be mistakenly used from other applications. This is due to the Android application developer's lack of secure coding knowledge. Depending on the content of functions that can be used by other applications, there are cases that may be troublesome if used by malware. For this reason, proper protection is required so that the functions can be used only from intended applications.

In the sample code of this guide, access definition is in the <queries> element for the sample that sends queries to or operates other applications. This complies with the principle of least privilege introduced for the package access specification for Android 11 .

Also, Google Play restricts the use of high risk or sensitive information permissions, including the SMS or call log permission groups, and applications that fail to meet policy requirements or submit a declaration form may be removed from Google Play[1] .

3.1.2 "Threats": Attacks that Threaten Assets

In the previous section, we discussed assets in Android smartphones. This section describes the threats to those assets, in other words, the attacks that threaten them. Simply put, an asset is threatened when information assets are referenced, modified, deleted, or created by others without permission, or when functional assets are used by others without permission. Attacks that directly or indirectly manipulate such assets are called threats. The person or thing that carries out the attack is called the threat source. Attackers and malware are threat sources, not threats. We refer to the attacking behavior of the attacker or malware as the threat. The relationship between these terms is shown in Fig. 3.1.2.

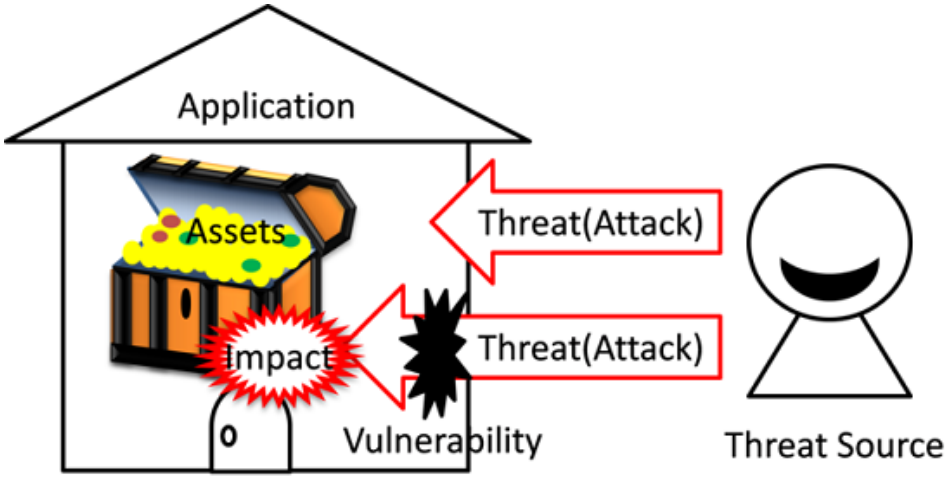


Fig. 3.1.2: Relationship between Assets, Threats, Threat Sources, Vulnerabilities, and Damage

Fig. 3.1.3 shows the general environment in which an Android application runs. In the following sections, we will use this figure as a basis for explaining threats to Android applications, so we will first explain how to view this figure.

[1] https://support.google.com/googleplay/android-developer/answer/9047303?hl=en&ref_topic=2364761

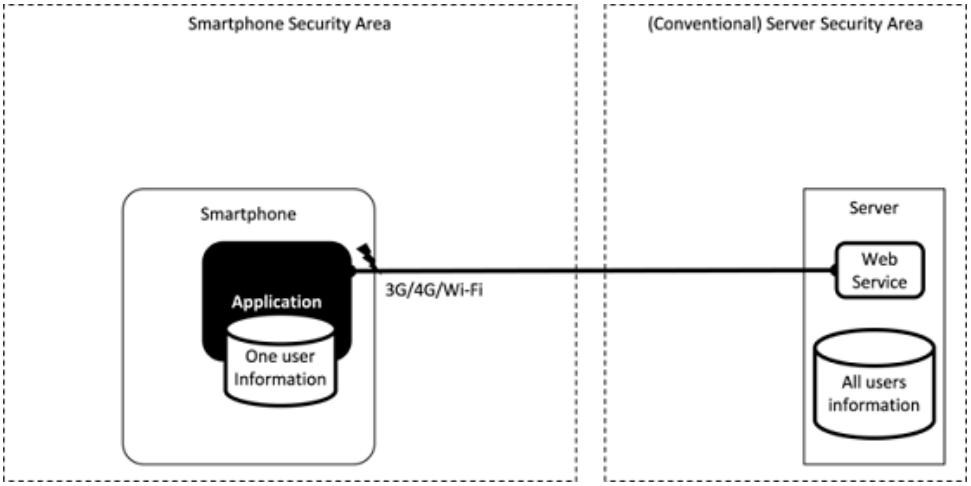


Fig. 3.1.3: Android General Environment in which an Android Application Runs

Smartphones and servers are placed on the left and right sides in the figure. Smartphones and servers communicate via 3G/4G/5G/Wi-Fi and the Internet. Although there are multiple applications in a smartphone, this figure focuses on a single application in order to explain the threats related to one application in the subsequent explanations. The application on a smartphone mainly handles the information of its user, while the web service on a server represents the centralized management of all users' information. Therefore, server security remains as important as before. Server security is not covered in this Guidebook because it is outside its scope.

In the following sections, we will use this diagram to explain the threats in Android applications.

3.1.2.1 Network-based Third-Party

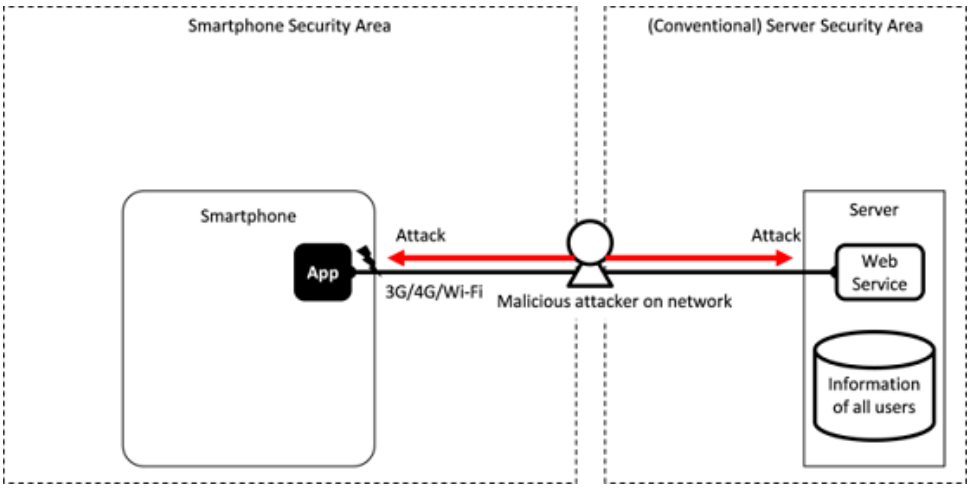


Fig. 3.1.4: Network-Based Malicious Third Party Attacking an Application

Generally, a smartphone application manages user information on a server so the information assets will move between the networks connecting them. As indicated in Fig. 3.1.4, a network-based malicious third party may access (sniff) any information during this communication or try to change information (data manipulation). The malicious attacker in the middle (also referred to as "Man in The Middle") can also pretend to be the real server tricking the application. Without saying, network-based malicious third parties will usually try to attack the server as well.

3.1.2.2 Threat Due to User-Installed Malware

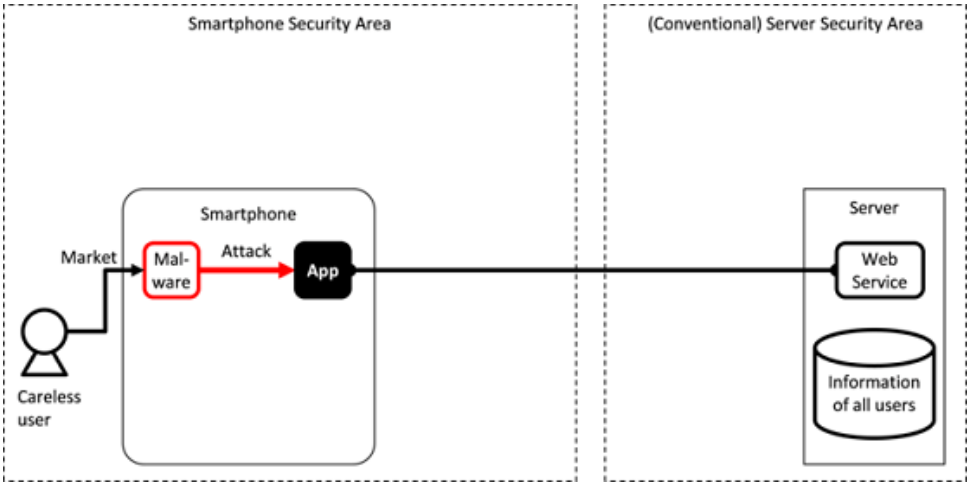


Fig. 3.1.5: Malware Installed by a User Attacks an Application

The biggest selling point of a smartphone is in its ability to acquire numerous applications from the market in order to expand on its features. The downside to users being able to freely install many applications is that they will sometimes mistakenly install malware. As shown in Fig. 3.1.5, malware may exploit the inter-application communication functions or a vulnerability in the application in order to gain access to information or function assets.

3.1.2.3 Threat of an Malicious File that Exploits a Vulnerability in an Application

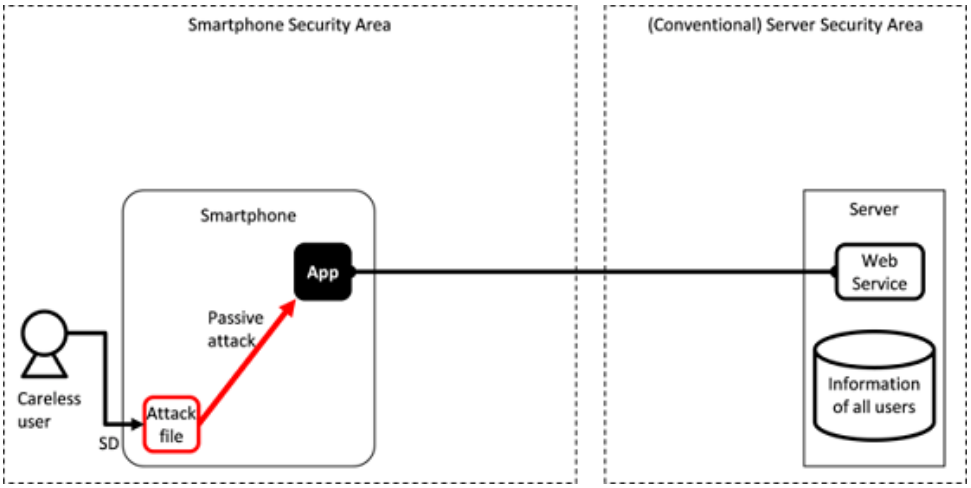


Fig. 3.1.6: Attack from Malicious Files that Exploit a Vulnerability in an Application

Various types of files such as music, images, videos and documents are widely available on the Internet and typically users will download many files to their SD card in order to use them on their smartphone. Furthermore, it is also common to download attached files sent in an e-mail. These files are later opened by a viewing or editing application.

If there is any vulnerability in the function of an application that processes these files, an attacker can use a malicious file to exploit it and gain access to information or function assets of the application. In particular, vulnerabilities are often present in processing a file format with a complex data structure. The attacker can fulfill many different goals when exploiting an application in this way.

As shown in Fig. 3.1.6, an attack file stays dormant until it is opened by a vulnerable application. Once it is opened, it will start causing havoc by taking advantage of an application's vulnerability. In comparison to an active attack, we

call this attack method a "Passive Attack."

3.1.2.4 Threats from a Malicious Smartphone User

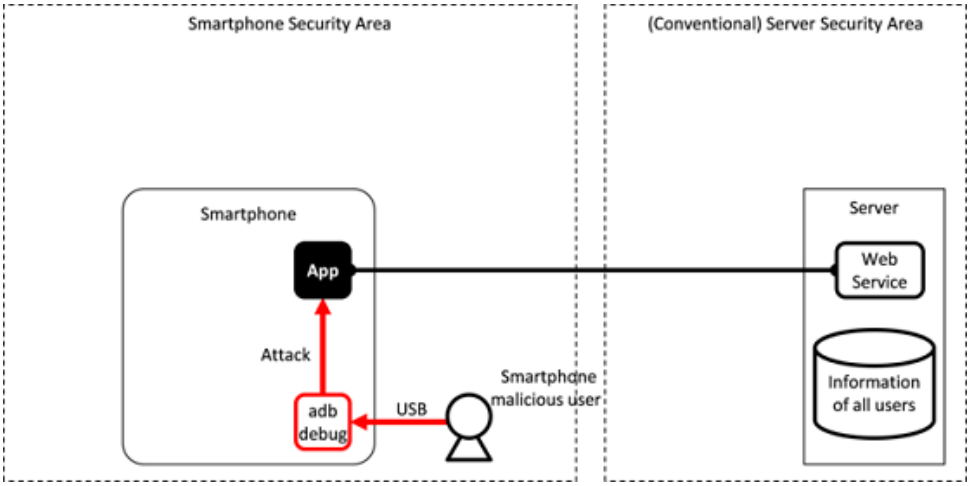


Fig. 3.1.7: Attacks from a Malicious Smartphone User

With regard to application development for an Android smartphone, the environment as well as features that help to develop and analyze an application are openly provided to the general user. Among the features that are provided, the useful ADB debugging feature can be accessed by anyone without registration or screening. This feature allows an Android smartphone user to easily perform OS or application analysis.

As it is shown in Fig. 3.1.7, a smartphone user with malicious intent can analyze an application by taking advantage of the debugging feature of ADB and try to gain access to information or function assets of an application. If the actual asset contained in the application belongs to the user, it poses no problem, but if the asset belongs to someone other than the user, such as the application developer, then it will become a concern. Accordingly, we need to be aware that the legitimate smartphone user can maliciously target the assets within an application.

3.1.2.5 Threats from Third Party in the Proximity of a Smartphone

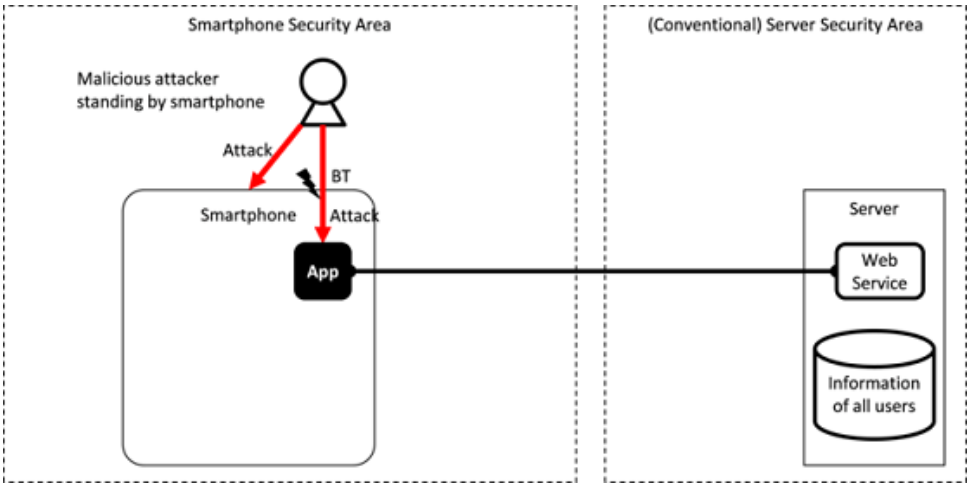


Fig. 3.1.8: Attacks from a Malicious Third Party in the Proximity of a Smartphone

Due to face that most smartphones possess a variety of near-field communication mechanisms, such as NFC, Bluetooth and Wi-Fi, we must not forget that attacks can occur from a malicious attacker who is in physical proximity

of a smartphone. An attacker can shoulder surf a password while peeping over a user who is inputting it in. Or, as indicated in Fig. 3.1.8, an attacker can be more sophisticated and attack the Bluetooth functionality of an application from a remote distance. There is also the threat that a malicious person could steal the smartphone creating a risk of data leakage or even destroy the smartphone causing a loss of critical information. Developers need to take these risks into consideration as well as early as the design stage.

3.1.2.6 Summary of Threats

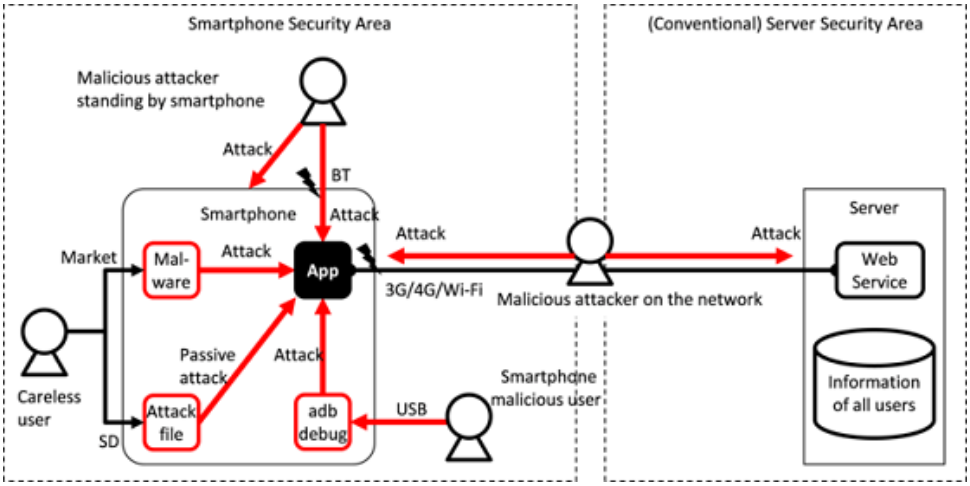


Fig. 3.1.9: Summary of the Various Attacks on Smartphone Applications

Fig. 3.1.9 summarizes the main types of threats explained in the previous sections. Smartphones are surrounded by a wide variety of threats and the figure above does not include all of them. Through our daily information gathering, we need to spread the awareness concerning the various threats that surround an Android application and be aware of them during the application's secure design and coding. The following literature that was created by Japan's Smartphone Security Association (JSSEC) contains other valuable information on the threats to smartphone security.

Security Guidebook for Using Smartphones and Tablets
https://www.jssec.org/dl/guidelines_v2.pdf
https://www.jssec.org/dl/guidelines2012Enew_v1.0.pdf (English)

Implementation Guidebook for Smartphone Network Security [Version 1]
https://www.jssec.org/dl/NetworkSecurityGuide1.pdf

Cloud Usage Guidebook for Business Purposes of Smartphones [Beta Version]
https://www.jssec.org/dl/cloudguide2012_beta.pdf

Guidebook for Reviewing the Implementation/Operation of MDM [Version 1]
https://www.jssec.org/dl/MDMGuideV1.pdf

3.1.3 Asset Classification and Protective Countermeasures

As was discussed in the previous sections, Android smartphones are surrounded by a variety of threats. Protecting every asset in an application from such threats could prove to be very difficult given the time it takes for development and due to technical limitations. Consequently, Android application developers should examine feasible countermeasures for their assets. This should be done according to priority level based on the developer's judgement criteria.

This is a subjective matter that is based on how the importance of an asset is viewed and what the accepted level of damage is.

In order to help decide on the protective countermeasures for each asset, we will classify them and stipulate the level of protective countermeasures for each group. This will be achieved by examining the legal basis, pertaining to the level of importance regarding the impact of any damages that can occur and the social responsibility of the developer (or organization). These will prove to be the judgement criteria when deciding on how to handle each asset and the implementation of the type of countermeasures. Since this will become a standard for application developers and organizations on determining how to handle an asset and provide protective countermeasures, it is necessary to specify the classification methods and pertaining countermeasures in accordance the application developer's (or organization's) circumstances.

Asset classification and protective countermeasure levels that are adopted in the Guidebook are shown below for reference:

Table 3.1.5: Asset Classification and Protective Countermeasure Levels

Asset Classification	Asset Level	Level of Protective Counter-Measures
High[2]	The amount of damage the asset causes is fatal and catastrophic to the organization or an individual's activity. i.e.) When an asset at this level is damaged, the organization will not be able to continue its business.	Provide protection against sophisticated attacks that break through the Android OS security model, such as attacks from a state where root privileges have been compromised, or attacks that alter the dex portion of an APK. Ensure security takes priority over other elements such as user experience, etc.
Medium	The amount of damage the asset causes has a substantial impact the organization or an individual's activity. i.e.) When an asset at this level is damaged, the organization's profit level deteriorates, adversely affecting its business.	Utilize the Android OS security model. It will provide protection covered under its scope. Ensure security takes priority over other elements such as user experience, etc.
Low	The amount of damage the asset causes has a limited impact on the organization or an individual's activity. i.e.) When an asset at this level is damaged, the organization's profit level will be affected but is able to compensate its losses from other resources.	Utilize the Android OS security model. It will provide protection covered under its scope. Compare security countermeasures with other elements such as user experience, etc. At this level, it is possible for non-security issues to take precedence over security issues.

Asset classification and protective countermeasures described in the Guidebook are proposed under the premise of a secure Android device where root privilege has not been compromised. Furthermore, it is based on the security measures that utilize the security model of Android OS. Specifically, we are hypothetically devising protective countermeasures by utilizing the Android OS security model on the premise of a functioning Android OS security model against assets that are classified lower than or equal to the medium level asset.

[2] We also believe in the necessity of protecting high level assets from attacks that are caused due the breaching of the Android OS security model. Such attacks include the compromise of root privileges and attacks that analyze or alter the APK binary. To protect these types of assets, we need to design sophisticated defensive countermeasures against such threats through the combination of multiple methods such as encryption, obfuscation, hardware support and server support. As the collection of know-how regarding these defenses cannot be easily written in this guidebook, and since appropriate defensive design differ in accordance to individual circumstances, we have deemed them to be outside of the Guidebook's scope. We recommend that you consult with a security specialist who is well versed in tamper resistant designs of Android if your device requires protection from sophisticated attacks that include attacks resulting from the compromise of root privileges or attacks caused by the analysis or alteration of an APK file.

3.1.4 Sensitive Information

The term "sensitive information", instead of information asset, will be used from now on in the Guidebook. As it has been aforementioned in the previous section, we have to determine the asset level and the level of protective countermeasures for each information asset that an application handles.

3.2 Handling Input Data Carefully and Securely

Validating input data is the easiest and yet most effective secure coding method. All data that is inputted into the application either directly or indirectly by an outside source needs to be properly validated. To illustrate best practices of input data validation, the following is an example of an Activity as used in a program that receives data from Intent.

It is possible that an Activity can receive data from an Intent that was tampered by an attacker. By sending data with a format or a value that a programmer is not expecting, the attacker can induce a malfunction in the application that leads to some sort of security incident. We must not forget that a user can become an attacker as well.

Intents are configured by action, data and extras, and we must be careful when accepting all forms of data that can be controlled by an attacker. We always need to validate the following items in any code that handles data from an untrusted source.

(a) Does the received data match the format that was expected by the programmer and does the value fall in the expected scope?

(b) Even if you have received the expected format and value, can you guarantee that the code which handles that data will not behave unexpectedly?

The next example is a simple sample where HTML is acquired from a remote web page in a designated URL and the code is displayed in TextView. However, there is a bug.

Sample Code that Displays HTML of a Remote Web page in TextView

```
TextView tv = (TextView) findViewById(R.id.textview);
InputStreamReader isr = null;
char[] text = new char[1024];
int read;
try {
    String urlstr = getIntent().getStringExtra("WEBPAGE_URL");
    URL url = new URL(urlstr);
    isr = new InputStreamReader(url.openConnection().getInputStream());
    while ((read=isr.read(text)) != -1) {
        tv.append(new String(text, 0, read));
    }
} catch (MalformedURLException e) { //...
```

From the viewpoint of (a), "urlstr is the correct URL", verified through the non-occurrence of a MalformedURLException by a new URL(). However, this is not sufficient. Furthermore, when a "file://..." formatted URL is designated by urlstr, the file of the internal file system is opened and is displayed in TextView rather than the remote web page. This does not fulfill the viewpoint of (b), since it does not guarantee the behavior which was expected by the programmer.

The next example shows a revision to fix the security bugs. Through the viewpoint of (a), the input data is validated by checking that "urlstr is a legitimate URL and the protocol is limited to http or https." As a result, even by the viewpoint of (b), the acquisition of an Internet-routed InputStream is guaranteed through url.openConnection().getInputStream().

Revised sample code that displays HTML of Internet-based Web page in TextView

```
TextView tv = (TextView) findViewById(R.id.textview);
InputStreamReader isr = null;
char[] text = new char[1024];
```

(continues on next page)

(continued from previous page)

```
int read;
try {
    String urlstr = getIntent().getStringExtra("WEBPAGE_URL");
    URL url = new URL(urlstr);
    String prot = url.getProtocol();
    if (!"http".equals(prot) && !"https".equals(prot)) {
        throw new MalformedURLException("invalid protocol");
    }
    isr = new InputStreamReader(url.openConnection().getInputStream());
    while ((read=isr.read(text)) != -1) {
        tv.append(new String(text, 0, read));
    }
} catch (MalformedURLException e) { //...
```

Validating the safety of input data is called "Input Validation" and it is a fundamental secure coding method. Surmising from the sense of the word of Input Validation, it is quite often the case where the viewpoint of (a) is heeded but the viewpoint of (b) is forgotten. It is important to remember that damage does not take place when data enters the program but when the program "uses" that data in an incorrect way. We hope that you will refer the URLs listed below.

The CERT Oracle Secure Coding Standard for Java
https://www.informit.com/store/cert-oracle-secure-coding-standard-for-java-9780321803955 (English)

SEI CERT Oracle Coding Standard for Java
https://wiki.sei.cmu.edu/confluence/display/java/SEI+CERT+Oracle+Coding+Standard+for+Java (English)

IPA "Secure Programming Course"
https://www.ipa.go.jp/security/awareness/vendor/programming/index.html (Japanese)

4 Using Technology in a Safe Way

In Android, there are many specific security related issues that pertain only to certain technologies such as Activities or SQLite. If a developer does not have enough knowledge about each of the different security issues regarding each technology when designing and coding, then unexpected vulnerabilities may arise. This chapter will explain about the different scenarios that developers will need to know when using their application components.

4.1 Creating/Using Activities

4.1.1 Sample Code

The risks and countermeasures of using Activities differ depending on how that Activity is being used. In this section, we have classified 4 types of Activities based on how the Activity is being used. You can find out which type of activity you are supposed to create through the following chart shown below. Since the secure coding best practice varies according to how the activity is used, we will also explain about the implementation of the Activity as well.

Table 4.1.1: Definition of Activity Types

Type	Definition
Private Activity	An activity that cannot be launched by another application, and therefore is the safest activity
Public Activity	An activity that is supposed to be used by an unspecified large number of applications.
Partner Activity	An activity that can only be used by specific applications made by a trusted partner company.
In-house Activity	An activity that can only be used by other in-house applications.

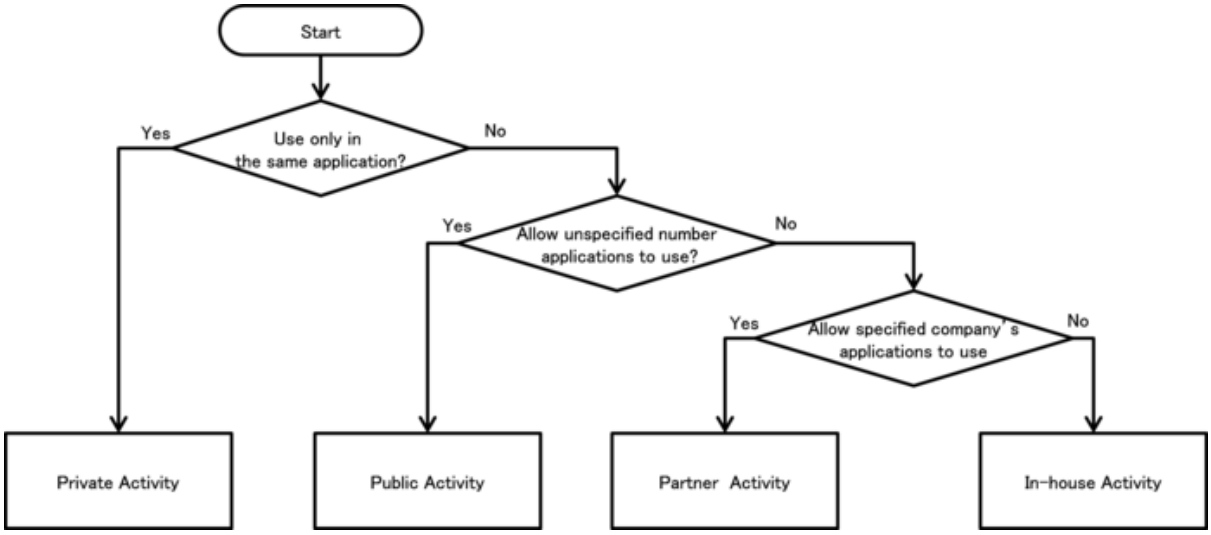


Fig. 4.1.1: Flow Figure to select Activity Type

4.1.1.1 Creating/Using Private Activities

Private Activities are Activities which cannot be launched by the other applications and therefore it is the safest Activity.

When using Activities that are only used within the application (Private Activity), as long as you use explicit Intents to the class then you do not have to worry about accidently sending it to any other application. However, there is a risk that a third party application can read an Intent that is used to start the Activity. Therefore it is necessary to make sure that if you are putting sensitive information inside an Intent used to start an Activity that you take countermeasures to make sure that it cannot be read by a malicious third party.

Sample code of how to create a Private Activity is shown below.

Points (Creating an Activity):

1. Do not specify taskAffinity.
2. Do not specify launchMode.
3. Explicitly set the exported attribute to false.
4. Handle the received intent carefully and securely, even though the intent was sent from the same application.
5. Sensitive information can be sent since it is sending and receiving all within the same application.

To make the Activity private, set the "exported" attribute of the Activity element in the AndroidManifest.xml to false.

```
AndroidManifest.xml
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
          >

  <application
      android:allowBackup="false"
      android:icon="@drawable/ic_launcher"
      android:label="@string/app_name" >

    <!-- Private activity -->
    <!-- *** POINT 1 *** Do not specify taskAffinity -->
    <!-- *** POINT 2 *** Do not specify launchMode -->
    <!-- *** POINT 3 *** Explicitly set the exported attribute to false. -->
    <activity
```

(continues on next page)

(continued from previous page)

```
        android:name=".PrivateActivity"
        android:label="@string/app_name"
        android:exported="false" />

    <!-- Public activity launched by launcher -->
    <activity
        android:name=".PrivateUserActivity"
        android:label="@string/app_name"
        android:exported="true" >
      <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER" />
      </intent-filter>
    </activity>
  </application>
</manifest>
```

PrivateActivity.java

```
/*
 * Copyright (C) 2012-2026 Japan Smartphone Security Association
 *
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 *
 *      http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */

package org.jssec.android.activity.privateactivity;

import android.app.Activity;
import android.content.Intent;
import android.os.Bundle;
import android.view.View;
import android.widget.Toast;

public class PrivateActivity extends Activity {

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.private_activity);

        // *** POINT 4 *** Handle the received Intent carefully and securely,
        // even though the Intent was sent from the same application.
        // Omitted, since this is a sample. Please refer to
        // "3.2 Handling Input Data Carefully and Securely."
        String param = getIntent().getStringExtra("PARAM");
        Toast.makeText(this,
                       String.format("Received param: \"%s\"", param),
```

(continues on next page)

(continued from previous page)

```
                        Toast.LENGTH_LONG).show();
    }

    public void onReturnResultClick(View view) {

        // *** POINT 5 *** Sensitive information can be sent since it is sending
        // and receiving all within the same application.
        Intent intent = new Intent();
        intent.putExtra("RESULT", "Sensitive Info");
        setResult(RESULT_OK, intent);
        finish();
    }
}
```

Next, we show the sample code for how to use the Private Activity.

Point (Using an Activity):

6. Do not set the FLAG_ACTIVITY_NEW_TASK flag for intents to start an activity.
7. Use the explicit Intents with the class specified to call an activity in the same application.
8. Sensitive information can be sent only by putExtra() since the destination activity is in the same application[1].
9. Handle the received result data carefully and securely, even though the data comes from an activity within the same application.

PrivateUserActivity.java

```
/*
 * Copyright (C) 2012-2026 Japan Smartphone Security Association
 *
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 *
 *      http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */

package org.jssec.android.activity.privateactivity;

import android.app.Activity;
import android.content.Intent;
import android.os.Bundle;
import android.view.View;
import android.widget.Toast;

public class PrivateUserActivity extends Activity {

    private static final int REQUEST_CODE = 1;

    @Override
```

(continues on next page)

[1] Caution: Unless points 1, 2 and 6 are abided by, there is a risk that Intents may be read by a third party. Please refer to 4.1.2.2. and 4.1.2.3. for more details.

(continued from previous page)

```
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.user_activity);
    }

    public void onUseActivityClick(View view) {

        // *** POINT 6 *** Do not set the FLAG_ACTIVITY_NEW_TASK flag
        // for intents to start an activity.
        // *** POINT 7 *** Use the explicit Intents with the class
        // specified to call an activity in the same application.
        Intent intent = new Intent(this, PrivateActivity.class);

        // *** POINT 8 *** Sensitive information can be sent only by putExtra()
        // since the destination activity is in the same application.
        intent.putExtra("PARAM", "Sensitive Info");

        startActivityForResult(intent, REQUEST_CODE);
    }

    @Override
    public void onActivityResult(int requestCode, int resultCode, Intent data) {
        super.onActivityResult(requestCode, resultCode, data);

        if (resultCode != RESULT_OK) return;

        switch (requestCode) {
        case REQUEST_CODE:
            String result = data.getStringExtra("RESULT");

            // *** POINT 9 *** Handle the received data carefully and securely,
            // even though the data comes from an activity within the same
            // application.
            // Omitted, since this is a sample. Please refer to
            // "3.2 Handling Input Data Carefully and Securely."
            Toast.makeText(this,
                           String.format("Received result: \"%s\"", result),
                           Toast.LENGTH_LONG).show();
            break;
        }
    }
}
```

4.1.1.2 Creating/Using Public Activities

Public Activities are Activities which are supposed to be used by an unspecified large number of applications. It is necessary to be aware that Public Activities may receive Intents sent from malware.

In addition, when using Public Activities, it is necessary to be aware of the fact that malware can also receive or read the Intents sent to them.

The sample code to create a Public Activity is shown below.

Points (Creating an Activity):

1. Explicitly set the exported attribute to true.
2. Handle the received intent carefully and securely.

3. When returning a result, do not include sensitive information.

AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
          >

  <application
      android:allowBackup="false"
      android:icon="@drawable/ic_launcher"
      android:label="@string/app_name" >

    <!-- Public Activity -->
    <!-- *** POINT 1 *** Explicitly set the exported attribute to true. -->
    <activity
        android:name=".PublicActivity"
        android:label="@string/app_name"
        android:exported="true">

      <!-- Define intent filter to receive an implicit intent for a specified␣
↪action -->
      <intent-filter>
        <action android:name="org.jssec.android.activity.MY_ACTION" />
        <category android:name="android.intent.category.DEFAULT" />
      </intent-filter>
    </activity>
  </application>
</manifest>
```

PublicActivity.java

```
/*
 * Copyright (C) 2012-2026 Japan Smartphone Security Association
 *
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 *
 *      http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */

package org.jssec.android.activity.publicactivity;

import android.app.Activity;
import android.content.Intent;
import android.os.Bundle;
import android.view.View;
import android.widget.Toast;

public class PublicActivity extends Activity {

    @Override
```

(continues on next page)

(continued from previous page)

```
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);

        // *** POINT 2 *** Handle the received intent carefully and securely.
        // Since this is a public activity, it is possible that the sending
        // application may be malware.
        // Omitted, since this is a sample. Please refer to
        // "3.2 Handling Input Data Carefully and Securely."
        String param = getIntent().getStringExtra("PARAM");
        Toast.makeText(this,
                       String.format("Received param: \"%s\"", param),
                       Toast.LENGTH_LONG).show();
    }

    public void onReturnResultClick(View view) {

        // *** POINT 3 *** When returning a result, do not include sensitive
        // information.
        // Since this is a public activity, it is possible that the receiving
        // application may be malware.
        // If there is no problem if the data gets received by malware,
        // then it can be returned as a result.
        Intent intent = new Intent();
        intent.putExtra("RESULT", "Not Sensitive Info");
        setResult(RESULT_OK, intent);
        finish();
    }
}
```

Next, Herein after sample code of Public Activity user side.

Points (Using an Activity):

4. Do not send sensitive information.
5. When receiving a result, handle the data carefully and securely.

PublicUserActivity.java

```
/*
 * Copyright (C) 2012-2026 Japan Smartphone Security Association
 *
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 *
 *      http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */

package org.jssec.android.activity.publicuser;

import android.app.Activity;
```

(continues on next page)

(continued from previous page)

```
import android.content.ActivityNotFoundException;
import android.content.Intent;
import android.os.Bundle;
import android.view.View;
import android.widget.Toast;

public class PublicUserActivity extends Activity {

    private static final int REQUEST_CODE = 1;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
    }

    public void onUseActivityClick(View view) {

        try {
            // *** POINT 4 *** Do not send sensitive information.
            Intent intent = new Intent("org.jssec.android.activity.MY_ACTION");
            intent.putExtra("PARAM", "Not Sensitive Info");
            startActivityForResult(intent, REQUEST_CODE);
        } catch (ActivityNotFoundException e) {
            Toast.makeText(this,
                           "Target activity not found.", Toast.LENGTH_LONG).show();
        }
    }

    @Override
    public void onActivityResult(int requestCode, int resultCode, Intent data) {
        super.onActivityResult(requestCode, resultCode, data);

        // *** POINT 5 *** When receiving a result, handle the data carefully and
        // securely.
        // Omitted, since this is a sample. Please refer to
        // "3.2 Handling Input Data Carefully and Securely."
        if (resultCode != RESULT_OK) return;
        switch (requestCode) {
        case REQUEST_CODE:
            String result = data.getStringExtra("RESULT");
            Toast.makeText(this,
                           String.format("Received result: \"%s\"", result),
                           Toast.LENGTH_LONG).show();
            break;
        }
    }
}
```

4.1.1.3 Creating/Using Partner Activities

Partner activities are Activities that can only be used by specific applications. They are used between cooperating partner companies that want to securely share information and functionality.

There is a risk that a third party application can read an Intent that is used to start the Activity. Therefore it is necessary to make sure that if you are putting sensitive information inside an Intent used to start an Activity that you

take countermeasures to make sure that it cannot be read by a malicious third party

Sample code for creating a Partner Activity is shown below.

Points (Creating an Activity):

1. Do not specify taskAffinity.
2. Do not specify launchMode.
3. Do not define the intent filter and explicitly set the exported attribute to true.
4. Verify the requesting application's certificate through a predefined whitelist.
5. Handle the received intent carefully and securely, even though the intent was sent from a partner application.
6. Only return Information that is granted to be disclosed to a partner application.

Please refer to "4.1.3.2. *Validating the Requesting Application*" for how to validate an application by a white list. Also, please refer to "5.2.1.3. *How to Verify the Hash Value of an Application's Certificate*" for how to verify the certificate hash value of a destination application which is specified in the whitelist.

AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
          >

  <application
      android:allowBackup="false"
      android:icon="@drawable/ic_launcher"
      android:label="@string/app_name" >

    <!-- Partner activity -->
    <!-- *** POINT 1 *** Do not specify taskAffinity -->
    <!-- *** POINT 2 *** Do not specify launchMode -->
    <!-- *** POINT 3 *** Do not define the intent filter and explicitly set the␣
↪exported attribute to true -->
    <activity
        android:name=".PartnerActivity"
        android:exported="true" />

  </application>
</manifest>
```

PartnerActivity.java

```
/*
 * Copyright (C) 2012-2026 Japan Smartphone Security Association
 *
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 *
 *      http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */

package org.jssec.android.activity.partneractivity;
```

(continues on next page)

(continued from previous page)

```
import org.jssec.android.shared.PkgCertWhitelists;
import org.jssec.android.shared.Utils;

import android.app.Activity;
import android.content.Context;
import android.content.Intent;
import android.os.Bundle;
import android.view.View;
import android.widget.Toast;

public class PartnerActivity extends Activity {

    // *** POINT 4 *** Verify the requesting application's certificate
    // through a predefined whitelist.
    private static PkgCertWhitelists sWhitelists = null;
    private static void buildWhitelists(Context context) {
        boolean isdebug = Utils.isDebuggable(context);
        sWhitelists = new PkgCertWhitelists();

        // Register certificate hash value of partner application
        // org.jssec.android.activity.partneruser.
        sWhitelists.add("org.jssec.android.activity.partneruser", isdebug ?
            // Certificate hash value of "androiddebugkey" in the debug.keystore.
            "0EFB7236 328348A9 89718BAD DF57F544 D5CCB4AE B9DB34BC 1E29DD26␣
↪F77C8255" :
            // Certificate hash value of "partner key" in the keystore.
            "1F039BB5 7861C27A 3916C778 8E78CE00 690B3974 3EB8259F E2627B8D␣
↪4C0EC35A");

        // Register the other partner applications in the same way.
    }
    private static boolean checkPartner(Context context, String pkgname) {
        if (sWhitelists == null) buildWhitelists(context);
        return sWhitelists.test(context, pkgname);
    }

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);

        // *** POINT 4 *** Verify the requesting application's certificate
        // through a predefined whitelist.
        if (!checkPartner(this, getCallingActivity().getPackageName())) {
            Toast.makeText(this,
                           "Requesting application is not a partner application.",
                           Toast.LENGTH_LONG).show();
            finish();
            return;
        }

        // *** POINT 5 *** Handle the received intent carefully and securely,
        // even though the intent was sent from a partner application.
        // Omitted, since this is a sample. Refer to
        // "3.2 Handling Input Data Carefully and Securely."
```

(continues on next page)

(continued from previous page)

```
        Toast.makeText(this, "Accessed by Partner App", Toast.LENGTH_LONG).show();
    }

    public void onReturnResultClick(View view) {

        // *** POINT 6 *** Only return Information that is granted to be disclosed
        // to a partner application.
        Intent intent = new Intent();
        intent.putExtra("RESULT", "Information for partner applications");
        setResult(RESULT_OK, intent);
        finish();
    }
}
```

PkgCertWhitelists.java

```
/*
 * Copyright (C) 2012-2026 Japan Smartphone Security Association
 *
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 *
 *      http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */

package org.jssec.android.shared;

import android.content.pm.PackageManager;
import java.util.HashMap;
import java.util.Map;
import android.content.Context;
import android.os.Build;

import static android.content.pm.PackageManager.CERT_INPUT_SHA256;

public class PkgCertWhitelists {
    private Map<String, String> mWhitelists = new HashMap<String, String>();

    public boolean add(String pkgname, String sha256) {
        if (pkgname == null) return false;
        if (sha256 == null) return false;

        sha256 = sha256.replaceAll(" ", "");
        if (sha256.length() != 64)
            return false;  // SHA-256 -> 32 bytes -> 64 chars
        sha256 = sha256.toUpperCase();
        if (sha256.replaceAll("[0-9A-F]+", "").length() != 0)
            return false;  // found non hex char

        mWhitelists.put(pkgname, sha256);
```

(continues on next page)

(continued from previous page)

```
        return true;
    }

    public boolean test(Context ctx, String pkgname) {
        // Get the correct hash value which corresponds to pkgname.
        String correctHash = mWhitelists.get(pkgname);

        // Compare the actual hash value of pkgname with the correct hash value.
        if (Build.VERSION.SDK_INT >= 28) {
            // ** if API Level >= 28, direct checking is possible
            PackageManager pm = ctx.getPackageManager();
            return pm.hasSigningCertificate(pkgname,
                                            Utils.hex2Bytes(correctHash),
                                            CERT_INPUT_SHA256);
        } else {
            // else use the facility of PkgCert
            return PkgCert.test(ctx, pkgname, correctHash);
        }
    }
}
```

PkgCert.java

```
/*
 * Copyright (C) 2012-2026 Japan Smartphone Security Association
 *
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 *
 *      http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */

package org.jssec.android.shared;

import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;

import android.content.Context;
import android.content.pm.PackageInfo;
import android.content.pm.PackageManager;
import android.content.pm.PackageManager.NameNotFoundException;
import android.content.pm.Signature;

public class PkgCert {

    public static boolean test(Context ctx, String pkgname, String correctHash) {
        if (correctHash == null) return false;
        correctHash = correctHash.replaceAll(" ", "");
        return correctHash.equals(hash(ctx, pkgname));
    }
```

(continues on next page)

(continued from previous page)

```
    public static String hash(Context ctx, String pkgname) {
        if (pkgname == null) return null;
        try {
            PackageManager pm = ctx.getPackageManager();
            PackageInfo pkginfo =
                pm.getPackageInfo(pkgname, PackageManager.GET_SIGNATURES);
            // Will not handle multiple signatures.
            if (pkginfo.signatures.length != 1) return null;
            Signature sig = pkginfo.signatures[0];
            byte[] cert = sig.toByteArray();
            byte[] sha256 = computeSha256(cert);
            return byte2hex(sha256);
        } catch (NameNotFoundException e) {
            return null;
        }
    }

    private static byte[] computeSha256(byte[] data) {
        try {
            return MessageDigest.getInstance("SHA-256").digest(data);
        } catch (NoSuchAlgorithmException e) {
            return null;
        }
    }

    private static String byte2hex(byte[] data) {
        if (data == null) return null;
        final StringBuilder hexadecimal = new StringBuilder();
        for (final byte b : data) {
            hexadecimal.append(String.format("%02X", b));
        }
        return hexadecimal.toString();
    }
}
```

Sample code for using a Partner Activity is described below.

Points (Using an Activity):

7. Verify if the certificate of the target application has been registered in a whitelist.
8. Do not set the FLAG_ACTIVITY_NEW_TASK flag for the intent that start an activity.
9. Only send information that is granted to be disclosed to a Partner Activity only by putExtra().
10. Use explicit intent to call a Partner Activity.
11. Use startActivityForResult() to call a Partner Activity.
12. Handle the received result data carefully and securely, even though the data comes from a partner application.

Refer to "4.1.3.2. *Validating the Requesting Application*" for how to validate applications by white list. Also please refer to "5.2.1.3. *How to Verify the Hash Value of an Application's Certificate*" for how to verify the certificate hash value of a destination application which is to be specified in a white list.

```
AndroidManifest.xml
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
          >
```

(continues on next page)

(continued from previous page)

```
  <queries>
    <package android:name="org.jssec.android.activity.partneractivity" />
  </queries>

  <application
      android:allowBackup="false"
      android:icon="@drawable/ic_launcher"
      android:label="@string/app_name" >

    <activity
        android:name=".PartnerUserActivity"
        android:label="@string/app_name"
        android:exported="true" >
      <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER" />
      </intent-filter>
    </activity>
  </application>
</manifest>
```

PartnerUserActivity.java

```
/*
 * Copyright (C) 2012-2026 Japan Smartphone Security Association
 *
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 *
 *      http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */

package org.jssec.android.activity.partneruser;

import org.jssec.android.shared.PkgCertWhitelists;
import org.jssec.android.shared.Utils;

import android.app.Activity;
import android.content.ActivityNotFoundException;
import android.content.Context;
import android.content.Intent;
import android.os.Bundle;
import android.view.View;
import android.widget.Toast;

public class PartnerUserActivity extends Activity {

    // *** POINT 7 *** Verify if the certificate of a target application
    // has been registered in a white list.
```

(continues on next page)

(continued from previous page)

```
    private static PkgCertWhitelists sWhitelists = null;
    private static void buildWhitelists(Context context) {
        boolean isdebug = Utils.isDebuggable(context);
        sWhitelists = new PkgCertWhitelists();

        // Register the certificate hash value of partner application
        // org.jssec.android.activity.partneractivity.
        sWhitelists.add("org.jssec.android.activity.partneractivity", isdebug ?
            // Certificate hash value of "androiddebugkey" is in debug.keystore.
            "0EFB7236 328348A9 89718BAD DF57F544 D5CCB4AE B9DB34BC 1E29DD26␣
↪F77C8255" :
            // Certificate hash value of "my company key" is in the keystore.
            "D397D343 A5CBC10F 4EDDEB7C A10062DE 5690984F 1FB9E88B D7B3A7C2␣
↪42E142CA");

        // Register the other partner applications in the same way.
    }
    private static boolean checkPartner(Context context, String pkgname) {
        if (sWhitelists == null) buildWhitelists(context);
        return sWhitelists.test(context, pkgname);
    }

    private static final int REQUEST_CODE = 1;

    // Information related the target partner activity
    private static final String TARGET_PACKAGE =
            "org.jssec.android.activity.partneractivity";
    private static final String TARGET_ACTIVITY =
            "org.jssec.android.activity.partneractivity.PartnerActivity";

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
    }

    public void onUseActivityClick(View view) {

        // *** POINT 7 *** Verify if the certificate of the target application
        // has been registered in the own white list.
        if (!checkPartner(this, TARGET_PACKAGE)) {
            Toast.makeText(this,
                           "Target application is not a partner application.",
                           Toast.LENGTH_LONG).show();
            return;
        }

        try {
            // *** POINT 8 *** Do not set the FLAG_ACTIVITY_NEW_TASK flag for
            // the intent that start an activity.
            Intent intent = new Intent();

            // *** POINT 9 *** Only send information that is granted to be
            // disclosed to a Partner Activity only by putExtra().
            intent.putExtra("PARAM", "Info for Partner Apps");
```

(continues on next page)

(continued from previous page)

```
            // *** POINT 10 *** Use explicit intent to call a Partner Activity.
            intent.setClassName(TARGET_PACKAGE, TARGET_ACTIVITY);

            // *** POINT 11 *** Use startActivityForResult() to call a Partner
            // Activity.
            startActivityForResult(intent, REQUEST_CODE);
        }
        catch (ActivityNotFoundException e) {
            Toast.makeText(this,
                           "Target activity not found.",
                           Toast.LENGTH_LONG).show();
        }
    }

    @Override
    public void onActivityResult(int requestCode, int resultCode, Intent data) {
        super.onActivityResult(requestCode, resultCode, data);

        if (resultCode != RESULT_OK) return;

        switch (requestCode) {
        case REQUEST_CODE:
            String result = data.getStringExtra("RESULT");

            // *** POINT 12 *** Handle the received data carefully and securely,
            // even though the data comes from a partner application.
            // Omitted, since this is a sample. Please refer to
            // "3.2 Handling Input Data Carefully and Securely."
            Toast.makeText(this,
                           String.format("Received result: \"%s\"", result),
                           Toast.LENGTH_LONG).show();
            break;
        }
    }
}
```

PkgCertWhitelists.java

```
/*
 * Copyright (C) 2012-2026 Japan Smartphone Security Association
 *
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 *
 *      http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */

package org.jssec.android.shared;

import android.content.pm.PackageManager;
```

(continues on next page)

(continued from previous page)

```
import java.util.HashMap;
import java.util.Map;
import android.content.Context;
import android.os.Build;

import static android.content.pm.PackageManager.CERT_INPUT_SHA256;

public class PkgCertWhitelists {
    private Map<String, String> mWhitelists = new HashMap<String, String>();

    public boolean add(String pkgname, String sha256) {
        if (pkgname == null) return false;
        if (sha256 == null) return false;

        sha256 = sha256.replaceAll(" ", "");
        if (sha256.length() != 64)
            return false;  // SHA-256 -> 32 bytes -> 64 chars
        sha256 = sha256.toUpperCase();
        if (sha256.replaceAll("[0-9A-F]+", "").length() != 0)
            return false;  // found non hex char

        mWhitelists.put(pkgname, sha256);
        return true;
    }

    public boolean test(Context ctx, String pkgname) {
        // Get the correct hash value which corresponds to pkgname.
        String correctHash = mWhitelists.get(pkgname);

        // Compare the actual hash value of pkgname with the correct hash value.
        if (Build.VERSION.SDK_INT >= 28) {
            // ** if API Level >= 28, direct checking is possible
            PackageManager pm = ctx.getPackageManager();
            return pm.hasSigningCertificate(pkgname,
                                            Utils.hex2Bytes(correctHash),
                                            CERT_INPUT_SHA256);
        } else {
            // else use the facility of PkgCert
            return PkgCert.test(ctx, pkgname, correctHash);
        }
    }
}
```

PkgCert.java

```
/*
 * Copyright (C) 2012-2026 Japan Smartphone Security Association
 *
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 *
 *      http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
```

(continues on next page)

(continued from previous page)

```
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */

package org.jssec.android.shared;

import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;

import android.content.Context;
import android.content.pm.PackageInfo;
import android.content.pm.PackageManager;
import android.content.pm.PackageManager.NameNotFoundException;
import android.content.pm.Signature;

public class PkgCert {

    public static boolean test(Context ctx, String pkgname, String correctHash) {
        if (correctHash == null) return false;
        correctHash = correctHash.replaceAll(" ", "");
        return correctHash.equals(hash(ctx, pkgname));
    }

    public static String hash(Context ctx, String pkgname) {
        if (pkgname == null) return null;
        try {
            PackageManager pm = ctx.getPackageManager();
            PackageInfo pkginfo =
                pm.getPackageInfo(pkgname, PackageManager.GET_SIGNATURES);
            // Will not handle multiple signatures.
            if (pkginfo.signatures.length != 1) return null;
            Signature sig = pkginfo.signatures[0];
            byte[] cert = sig.toByteArray();
            byte[] sha256 = computeSha256(cert);
            return byte2hex(sha256);
        } catch (NameNotFoundException e) {
            return null;
        }
    }

    private static byte[] computeSha256(byte[] data) {
        try {
            return MessageDigest.getInstance("SHA-256").digest(data);
        } catch (NoSuchAlgorithmException e) {
            return null;
        }
    }

    private static String byte2hex(byte[] data) {
        if (data == null) return null;
        final StringBuilder hexadecimal = new StringBuilder();
        for (final byte b : data) {
            hexadecimal.append(String.format("%02X", b));
        }
        return hexadecimal.toString();
    }
```

(continues on next page)

(continued from previous page)

```
}
```

4.1.1.4 Creating/Using In-house Activities

In-house activities are the Activities which are prohibited to be used by applications other than other in-house applications. They are used in applications developed internally that want to securely share information and functionality.

There is a risk that a third party application can read an Intent that is used to start the Activity. Therefore it is necessary to make sure that if you are putting sensitive information inside an Intent used to start an Activity that you take countermeasures to make sure that it cannot be read by a malicious third party.

Sample code for creating an In-house Activity is shown below.

Points (Creating an Activity):

1. Define an in-house signature permission.
2. Do not specify taskAffinity.
3. Do not specify launchMode.
4. Require the in-house signature permission.
5. Do not define an intent filter and explicitly set the exported attribute to true.
6. Verify that the in-house signature permission is defined by an in-house application.
7. Handle the received intent carefully and securely, even though the intent was sent from an in-house application.
8. Sensitive information can be returned since the requesting application is in-house.
9. When exporting an APK, sign the APK with the same developer key as the requesting application.

```
AndroidManifest.xml
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
          >

  <!-- *** POINT 1 *** Define an in-house signature permission -->
  <permission
      android:name="org.jssec.android.activity.inhouseactivity.MY_PERMISSION"
      android:protectionLevel="signature" />

  <application
      android:allowBackup="false"
      android:icon="@drawable/ic_launcher"
      android:label="@string/app_name" >

    <!-- In-house Activity -->
    <!-- *** POINT 2 *** Do not specify taskAffinity -->
    <!-- *** POINT 3 *** Do not specify launchMode -->
    <!-- *** POINT 4 *** Require the in-house signature permission  -->
    <!-- *** POINT 5 *** Do not define the intent filter and explicitly set the␣
↪exported attribute to true -->
    <activity
        android:name="org.jssec.android.activity.inhouseactivity.InhouseActivity"
        android:exported="true"
        android:permission="org.jssec.android.activity.inhouseactivity.MY_
↪PERMISSION" />
  </application>
</manifest>
```

InhouseActivity.java

```java
/*
 * Copyright (C) 2012-2026 Japan Smartphone Security Association
 *
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 *
 *      http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */

package org.jssec.android.activity.inhouseactivity;

import org.jssec.android.shared.SigPerm;
import org.jssec.android.shared.Utils;

import android.app.Activity;
import android.content.Context;
import android.content.Intent;
import android.os.Bundle;
import android.view.View;
import android.widget.Toast;

public class InhouseActivity extends Activity {

    // In-house Signature Permission
    private static final String MY_PERMISSION =
            "org.jssec.android.activity.inhouseactivity.MY_PERMISSION";

    // In-house certificate hash value
    private static String sMyCertHash = null;
    private static String myCertHash(Context context) {
        if (sMyCertHash == null) {
            if (Utils.isDebuggable(context)) {
                // Certificate hash value of "androiddebugkey" in the
                // debug.keystore.
                sMyCertHash = "0EFB7236 328348A9 89718BAD DF57F544 D5CCB4AE␣
↪B9DB34BC 1E29DD26 F77C8255";
            } else {
                // Certificate hash value of "my company key" in the keystore.
                sMyCertHash = "D397D343 A5CBC10F 4EDDEB7C A10062DE 5690984F␣
↪1FB9E88B D7B3A7C2 42E142CA";
            }
        }
        return sMyCertHash;
    }

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
```

(continues on next page)

(continued from previous page)

```
        // *** POINT 6 *** Verify that the in-house signature permission is
        // defined by an in-house application.
        if (!SigPerm.test(this, MY_PERMISSION, myCertHash(this))) {
            Toast.makeText(this,
                           "The in-house signature permission is not declared by␣
↪in-house application.",
                           Toast.LENGTH_LONG).show();
            finish();
            return;
        }

        // *** POINT 7 *** Handle the received intent carefully and securely,
        // even though the intent was sent from an in-house application.
        // Omitted, since this is a sample. Please refer to
        // "3.2 Handling Input Data Carefully and Securely."
        String param = getIntent().getStringExtra("PARAM");
        Toast.makeText(this,
                       String.format("Received param: \"%s\"", param),
                       Toast.LENGTH_LONG).show();
    }

    public void onReturnResultClick(View view) {

        // *** POINT 8 *** Sensitive information can be returned since
        // the requesting application is in-house.
        Intent intent = new Intent();
        intent.putExtra("RESULT", "Sensitive Info");
        setResult(RESULT_OK, intent);
        finish();
    }
}
```

SigPerm.java

```
/*
 * Copyright (C) 2012-2026 Japan Smartphone Security Association
 *
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 *
 *      http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */

package org.jssec.android.shared;

import android.content.Context;
import android.content.pm.PackageManager;
import android.content.pm.PackageManager.NameNotFoundException;
import android.content.pm.PermissionInfo;
```

(continues on next page)

(continued from previous page)

```
import android.os.Build;

import static android.content.pm.PackageManager.CERT_INPUT_SHA256;

public class SigPerm {

    public static boolean test(Context ctx, String sigPermName,
                               String correctHash) {
        if (correctHash == null) return false;
        correctHash = correctHash.replaceAll(" ", "");
        try {
            // Get the package name of the application which declares a permission
            // named sigPermName.
            PackageManager pm = ctx.getPackageManager();
            PermissionInfo pi =
                pm.getPermissionInfo(sigPermName, PackageManager.GET_META_DATA);
            String pkgname = pi.packageName;
            // Fail if the permission named sigPermName is not a Signature
            // Permission
            if (pi.protectionLevel != PermissionInfo.PROTECTION_SIGNATURE)
                return false;

            // Compare the actual hash value of pkgname with the correct hash
            // value.
            if (Build.VERSION.SDK_INT >= 28) {
                // ** if API Level >= 28, direct check is possible
                return pm.hasSigningCertificate(pkgname,
                                                Utils.hex2Bytes(correctHash),
                                                CERT_INPUT_SHA256);
            } else {
                // else(API Level < 28) use the facility of PkgCert
                return correctHash.equals(PkgCert.hash(ctx, pkgname));
            }

        } catch (NameNotFoundException e) {
            return false;
        }
    }
}
```

PkgCert.java

```
/*
 * Copyright (C) 2012-2026 Japan Smartphone Security Association
 *
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 *
 *      http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */
```

(continues on next page)

(continued from previous page)

```
package org.jssec.android.shared;

import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;

import android.content.Context;
import android.content.pm.PackageInfo;
import android.content.pm.PackageManager;
import android.content.pm.PackageManager.NameNotFoundException;
import android.content.pm.Signature;

public class PkgCert {

    public static boolean test(Context ctx, String pkgname, String correctHash) {
        if (correctHash == null) return false;
        correctHash = correctHash.replaceAll(" ", "");
        return correctHash.equals(hash(ctx, pkgname));
    }

    public static String hash(Context ctx, String pkgname) {
        if (pkgname == null) return null;
        try {
            PackageManager pm = ctx.getPackageManager();
            PackageInfo pkginfo =
                pm.getPackageInfo(pkgname, PackageManager.GET_SIGNATURES);
            // Will not handle multiple signatures.
            if (pkginfo.signatures.length != 1) return null;
            Signature sig = pkginfo.signatures[0];
            byte[] cert = sig.toByteArray();
            byte[] sha256 = computeSha256(cert);
            return byte2hex(sha256);
        } catch (NameNotFoundException e) {
            return null;
        }
    }

    private static byte[] computeSha256(byte[] data) {
        try {
            return MessageDigest.getInstance("SHA-256").digest(data);
        } catch (NoSuchAlgorithmException e) {
            return null;
        }
    }

    private static String byte2hex(byte[] data) {
        if (data == null) return null;
        final StringBuilder hexadecimal = new StringBuilder();
        for (final byte b : data) {
            hexadecimal.append(String.format("%02X", b));
        }
        return hexadecimal.toString();
    }
}
```

*** Point9 *** When exporting an APK, sign the APK with the same developer key as the requesting application.

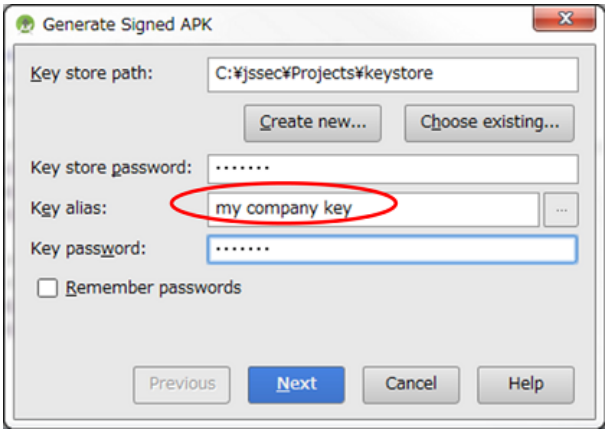


Fig. 4.1.2: Sign the APK with the same developer key as the requesting application

Sample code for using an In-house Activity is described below.

Points (Using an activity):

10. Declare that you want to use the in-house signature permission.
11. Verify that the in-house signature permission is defined by an in-house application.
12. Verify that the destination application is signed with the in-house certificate.
13. Sensitive information can be sent only by putExtra() since the destination application is in-house.
14. Use explicit intents to call an In-house Activity.
15. Handle the received data carefully and securely, even though the data came from an in-house application.
16. When exporting an APK, sign the APK with the same developer key as the destination application.

AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
          >

  <queries>
    <package android:name="org.jssec.android.activity.inhouseactivity" />
  </queries>

  <!-- *** POINT 10 *** Declare to use the in-house signature permission -->
  <uses-permission
      android:name="org.jssec.android.activity.inhouseactivity.MY_PERMISSION" />

  <application
      android:allowBackup="false"
      android:icon="@drawable/ic_launcher"
      android:label="@string/app_name" >

    <activity
        android:name="org.jssec.android.activity.inhouseuser.InhouseUserActivity"
        android:label="@string/app_name"
        android:exported="true" >
      <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER" />
      </intent-filter>
    </activity>
```

(continues on next page)

(continued from previous page)

```
  </application>
</manifest>
```

InhouseUserActivity.java

```
/*
 * Copyright (C) 2012-2026 Japan Smartphone Security Association
 *
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 *
 *      http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */

package org.jssec.android.activity.inhouseuser;

import org.jssec.android.shared.PkgCert;
import org.jssec.android.shared.SigPerm;
import org.jssec.android.shared.Utils;

import android.app.Activity;
import android.content.ActivityNotFoundException;
import android.content.Context;
import android.content.Intent;
import android.os.Bundle;
import android.view.View;
import android.widget.Toast;

public class InhouseUserActivity extends Activity {

    // Target Activity information
    private static final String TARGET_PACKAGE =
            "org.jssec.android.activity.inhouseactivity";
    private static final String TARGET_ACTIVITY =
            "org.jssec.android.activity.inhouseactivity.InhouseActivity";

    // In-house Signature Permission
    private static final String MY_PERMISSION =
            "org.jssec.android.activity.inhouseactivity.MY_PERMISSION";

    // In-house certificate hash value
    private static String sMyCertHash = null;
    private static String myCertHash(Context context) {
        if (sMyCertHash == null) {
            if (Utils.isDebuggable(context)) {
                // Certificate hash value of "androiddebugkey" in the
                // debug.keystore.
                sMyCertHash = "0EFB7236 328348A9 89718BAD DF57F544 D5CCB4AE␣
↪B9DB34BC 1E29DD26 F77C8255";
            } else {
```

(continues on next page)

(continued from previous page)

```
                // Certificate hash value of "my company key" in the keystore.
                sMyCertHash = "D397D343 A5CBC10F 4EDDEB7C A10062DE 5690984F␣
↪1FB9E88B D7B3A7C2 42E142CA";
            }
        }
        return sMyCertHash;
    }

    private static final int REQUEST_CODE = 1;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
    }

    public void onUseActivityClick(View view) {

        // *** POINT 11 *** Verify that the in-house signature permission is
        // defined by an in-house application.
        if (!SigPerm.test(this, MY_PERMISSION, myCertHash(this))) {
            Toast.makeText(this,
                           "The in-house signature permission is not declared by␣
↪in-house application.",
                           Toast.LENGTH_LONG).show();
            return;
        }

        // ** POINT 12 *** Verify that the destination application is signed
        // with the in-house certificate.
        if (!PkgCert.test(this, TARGET_PACKAGE, myCertHash(this))) {
            Toast.makeText(this,
                           "Target application is not an in-house application.",
                           Toast.LENGTH_LONG).show();
            return;
        }

        try {
            Intent intent = new Intent();

            // *** POINT 13 *** Sensitive information can be sent only by
            // putExtra() since the destination application is in-house.
            intent.putExtra("PARAM", "Sensitive Info");

            // *** POINT 14 *** Use explicit intents to call an In-house Activity.
            intent.setClassName(TARGET_PACKAGE, TARGET_ACTIVITY);
            startActivityForResult(intent, REQUEST_CODE);
        }
        catch (ActivityNotFoundException e) {
            Toast.makeText(this,
                           "Target activity not found.",
                           Toast.LENGTH_LONG).show();
        }
    }

    @Override
```

(continues on next page)

(continued from previous page)

```
    public void onActivityResult(int requestCode, int resultCode, Intent data) {
        super.onActivityResult(requestCode, resultCode, data);

        if (resultCode != RESULT_OK) return;

        switch (requestCode) {
        case REQUEST_CODE:
            String result = data.getStringExtra("RESULT");

            // *** POINT 15 *** Handle the received data carefully and securely,
            // even though the data came from an in-house application.
            // Omitted, since this is a sample. Please refer to
            // "3.2 Handling Input Data Carefully and Securely."
            Toast.makeText(this,
                           String.format("Received result: \"%s\"", result),
                           Toast.LENGTH_LONG).show();
            break;
        }
    }
}
```

SigPerm.java

```
/*
 * Copyright (C) 2012-2026 Japan Smartphone Security Association
 *
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 *
 *      http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */

package org.jssec.android.shared;

import android.content.Context;
import android.content.pm.PackageManager;
import android.content.pm.PackageManager.NameNotFoundException;
import android.content.pm.PermissionInfo;
import android.os.Build;

import static android.content.pm.PackageManager.CERT_INPUT_SHA256;

public class SigPerm {

    public static boolean test(Context ctx, String sigPermName,
                               String correctHash) {
        if (correctHash == null) return false;
        correctHash = correctHash.replaceAll(" ", "");
        try {
            // Get the package name of the application which declares a permission
```

(continues on next page)

(continued from previous page)

```
            // named sigPermName.
            PackageManager pm = ctx.getPackageManager();
            PermissionInfo pi =
                pm.getPermissionInfo(sigPermName, PackageManager.GET_META_DATA);
            String pkgname = pi.packageName;
            // Fail if the permission named sigPermName is not a Signature
            // Permission
            if (pi.protectionLevel != PermissionInfo.PROTECTION_SIGNATURE)
                return false;

            // Compare the actual hash value of pkgname with the correct hash
            // value.
            if (Build.VERSION.SDK_INT >= 28) {
                // ** if API Level >= 28, direct check is possible
                return pm.hasSigningCertificate(pkgname,
                                                Utils.hex2Bytes(correctHash),
                                                CERT_INPUT_SHA256);
            } else {
                // else(API Level < 28) use the facility of PkgCert
                return correctHash.equals(PkgCert.hash(ctx, pkgname));
            }

        } catch (NameNotFoundException e) {
            return false;
        }
    }
}
```

PkgCert.java

```
/*
 * Copyright (C) 2012-2026 Japan Smartphone Security Association
 *
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 *
 *      http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */

package org.jssec.android.shared;

import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;

import android.content.Context;
import android.content.pm.PackageInfo;
import android.content.pm.PackageManager;
import android.content.pm.PackageManager.NameNotFoundException;
import android.content.pm.Signature;
```

(continues on next page)

(continued from previous page)

```
public class PkgCert {

    public static boolean test(Context ctx, String pkgname, String correctHash) {
        if (correctHash == null) return false;
        correctHash = correctHash.replaceAll(" ", "");
        return correctHash.equals(hash(ctx, pkgname));
    }

    public static String hash(Context ctx, String pkgname) {
        if (pkgname == null) return null;
        try {
            PackageManager pm = ctx.getPackageManager();
            PackageInfo pkginfo =
                pm.getPackageInfo(pkgname, PackageManager.GET_SIGNATURES);
            // Will not handle multiple signatures.
            if (pkginfo.signatures.length != 1) return null;
            Signature sig = pkginfo.signatures[0];
            byte[] cert = sig.toByteArray();
            byte[] sha256 = computeSha256(cert);
            return byte2hex(sha256);
        } catch (NameNotFoundException e) {
            return null;
        }
    }

    private static byte[] computeSha256(byte[] data) {
        try {
            return MessageDigest.getInstance("SHA-256").digest(data);
        } catch (NoSuchAlgorithmException e) {
            return null;
        }
    }

    private static String byte2hex(byte[] data) {
        if (data == null) return null;
        final StringBuilder hexadecimal = new StringBuilder();
        for (final byte b : data) {
            hexadecimal.append(String.format("%02X", b));
        }
        return hexadecimal.toString();
    }
}
```

*** Point 16 *** When exporting an APK, sign the APK with the same developer key as the destination application.

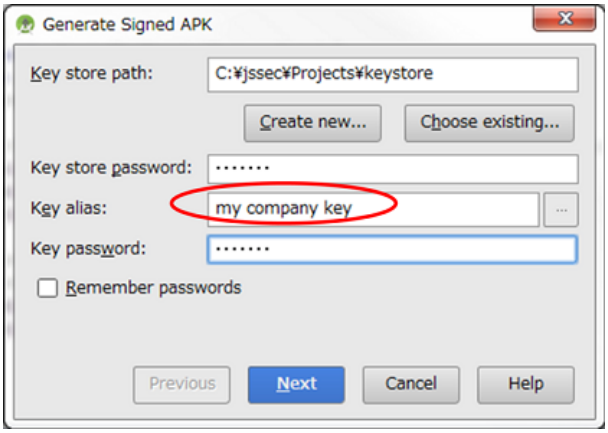


Fig. 4.1.3: Sign the APK with the same developer key as the destination application

4.1.2 Rule Book

Be sure to follow the rules below when creating or sending an Intent to an activity.

1. *Activities that are Used Only Internally to the Application Must be Set Private (Required)*
2. *Do Not Specify taskAffinity (Required)*
3. *Do Not Specify launchMode (Required)*
4. *Do Not Set the FLAG_ACTIVITY_NEW_TASK Flag for Intents that Start an Activity (Required)*
5. *Handling the Received Intent Carefully and Securely (Required)*
6. *Use an In-house Defined Signature Permission after Verifying that it is Defined by an In-House Application (Required)*
7. *When Returning a Result, Pay Attention to the Possibility of Information Leakage of that Result from the Destination Application (Required)*
8. *Use the explicit Intents if the destination Activity is predetermined. (Required)*
9. *Handle the Returned Data from a Requested Activity Carefully and Securely (Required)*
10. *Verify the Destination Activity if Linking with Another Company's Application (Required)*
11. *When Providing an Asset Secondhand, the Asset should be Protected with the Same Level of Protection (Required)*
12. *Sending Sensitive Information Should Be Limited as much as possible (Recommended)*

4.1.2.1 Activities that are Used Only Internally to the Application Must be Set Private (Required)

Activities which are only used in a single application are not required to be able to receive any Intents from other applications. Developers often assume that Activities intended to be private will not be attacked but it is necessary to explicitly make these Activities private in order to stop malicious Intents from being received.

```
AndroidManifest.xml
        <!-- Private activity -->
        <!-- *** 4.1.1.1 - POINT 3 *** Explicitly set the exported attribute to␣
↪false. -->
        <activity
            android:name=".PrivateActivity"
            android:label="@string/app_name"
            android:exported="false" />
```

Intent filters should not be set on activities that are only used in a single application. Due to the characteristics of Intent filters, Due to the characteristics of how Intent filters work, even if you intend to send an Intent to a Private Activity internally, if you send the Intent through an Intent filter than you may unintentionally start another Activity. Please see Advanced Topics "4.1.3.1. *Combination of Exported Attribute and Intent Filter Setting (For Activity)*" for more details.

```
AndroidManifest.xml(Not recommended)
       <!-- Private activity -->
       <!-- *** 4.1.1.1 - POINT 3 *** Explicitly set the exported attribute to␣
↪false. -->
       <activity
           android:name=".PictureActivity"
           android:label="@string/picture_name"
           android:exported="false" >
           <intent-filter>
               <action android:name=”org.jssec.android.activity.OPEN />
           </intent-filter>
       </activity>
```

4.1.2.2 Do Not Specify taskAffinity (Required)

In Android OS, Activities are managed by tasks. Task names are determined by the affinity that the root Activity has. On the other hand, for Activities other than root Activities, the task to which the Activity belongs is not determined by the Affinity only, but also depends on the Activity's launch mode. Please refer to "4.1.3.3. *About Root Activity*" for more details.

In the default setting, each Activity uses its package name as its affinity. As a result, tasks are allocated according to application, so all Activities in a single application will belong to the same task. To change the task allocation, you can make an explicit declaration for the affinity in the AndroidManifest.xml file or you can set a flag in an Intent sent to an Activity. However, if you change task allocations, there is a risk that another application could read the Intents sent to Activities belonging to another task.

Be sure not to specify android:taskAffinity in the AndroidManifest.xml file and use the default setting keeping the affinity as the package name in order to prevent sensitive information inside sent or received Intents from being read by another application.

Below is an example AndroidManifest.xml file for creating and using Private Activities.

```
AndroidManifest.xml
    <!-- *** 4.1.1.1 - POINT 1 *** Do not specify taskAffinity -->
    <application
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name" >

        <!-- *** 4.1.1.1 - POINT 1 *** Do not specify taskAffinity -->
        <activity
            android:name=".PrivateUserActivity"
            android:label="@string/app_name" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>

        <!-- Private activity -->
        <!-- *** 4.1.1.1 - POINT 1 *** Do not specify taskAffinity -->
        <activity
            android:name=".PrivateActivity"
```

(continues on next page)

(continued from previous page)

```
            android:label="@string/app_name"
            android:exported="false" />
    </application>
```

Please refer to the "Google Android Programming guide"[2], the Google Developer's API Guide "Tasks and Back Stack"[3], "4.1.3.3. *About Root Activity*" for more details about tasks and affinities.

4.1.2.3 Do Not Specify launchMode (Required)

The Activity launch mode is used to control the settings for creating new tasks and Activity instances when starting an Activity. By default it is set to "standard". In the "standard" setting, new instances are always created when starting an Activity, tasks follow the tasks belonging to the calling Activity, and it is not possible to create a new task. When a new task is created, it is possible for other applications to read the contents of the calling Intent so it is required to use the "standard" Activity launch mode setting when sensitive information is included in an Intent.

The Activity launch mode can be explicitly set in the android:launchMode attribute in the AndroidManifest.xml file, but because of the reason explained above, this should not be set in the Activity declaration and the value should be kept as the default "standard".

```
AndroidManifest.xml
        <!-- *** 4.1.1.1 - POINT 2 *** Do not specify launchMode -->
        <activity
            android:name=".PrivateUserActivity"
            android:label="@string/app_name" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>

        <!-- Private activity -->
        <!-- *** 4.1.1.1 - POINT 2 *** Do not specify launchMode -->
        <activity
            android:name=".PrivateActivity"
            android:label="@string/app_name"
            android:exported="false" />
    </application>
```

Please refer to "4.1.3.3. *About Root Activity*."

4.1.2.4 Do Not Set the FLAG_ACTIVITY_NEW_TASK Flag for Intents that Start an Activity (Required)

The launch mode of an Activity can be changed when executing startActivity() or startActivityForResult() and in some cases a new task may be generated. Therefore it is necessary to not change the launch mode of Activity during execution.

To change the Activity launch mode, set the Intent flags by using setFlags() or addFlags() and use that Intent as an argument to startActivity() or startActivityForResult(). FLAG_ACTIVITY_NEW_TASK is the flag used to create a new task. When the FLAG_ACTIVITY_NEW_TASK is set, a new task will be created if the called Activity does not exist in the background or foreground.

[2] Author Egawa, Fujii, Asano, Fujita, Yamada, Yamaoka, Sano, Takebata, "Google Android Programming Guide", ASCII Media Works, July 2009

[3] https://developer.android.com/guide/components/tasks-and-back-stack.html

The FLAG_ACTIVITY_MULTIPLE_TASK flag can be set simultaneously with FLAG_ACTIVITY_NEW_TASK. In this case, a new task will always be created. New tasks may be created with either setting so these should not be set with Intents that handle sensitive information.

Example of sending an intent

```
Intent intent = new Intent();

// * 4.1.1.1 - POINT 6 * Do not set the FLAG_ACTIVITY_NEW_TASK flag // for the intent to start an activity.

intent.setClass(this, PrivateActivity.class); intent.putExtra("PARAM", "Sensitive Info");

startActivityForResult(intent, REQUEST_CODE);
```

In addition, you may think that there is a way to prevent the contents of an Intent from being read even if a new task was created by explicitly setting the FLAG_ACTIVITY_EXCLUDE_FROM_RECENTS flag. However, even by using this method, the contents can be read by a third party so you should avoid any usage of FLAG_ACTIVITY_NEW_TASK.

Please refer to "4.1.3.1. *Combination of Exported Attribute and Intent Filter Setting (For Activity)*" and "4.1.3.3. *About Root Activity*".

4.1.2.5 Handling the Received Intent Carefully and Securely (Required)

Risks differ depending on the types of Activity, but when processing a received Intent data, the first thing you should do is input validation.

Since Public Activities can receive Intents from untrusted sources, they can be attacked by malware. On the other hand, Private Activities will never receive any Intents from other applications directly, but it is possible that a Public Activity in the targeted application may forward a malicious Intent to a Private Activity so you should not assume that Private Activities cannot receive any malicious input. Since Partner Activities and In-house Activities also have the risk of a malicious intent being forwarded to them as well, it is necessary to perform input validation on these Intents as well.

Please refer to "3.2. *Handling Input Data Carefully and Securely*".

4.1.2.6 Use an In-house Defined Signature Permission after Verifying that it is Defined by an In-House Application (Required)

Make sure to protect your in-house Activities by defining an in-house signature permission when creating the Activity. Since defining a permission in the AndroidManifest.xml file or declaring a permission request does not provide adequate security, please be sure to refer to "5.2.1.2. *How to Communicate Between In-house Applications with In-house-defined Signature Permission*."

4.1.2.7 When Returning a Result, Pay Attention to the Possibility of Information Leakage of that Result from the Destination Application (Required)

When you use setResult() to return data, the reliability of the destination application will depend on the Activity type. When Public Activities are used to return data, the destination may turn out to be malware in which case that information could be used in a malicious way. For Private and In-house Activities, there is not much need to worry about data being returned to be used maliciously because they are being returned to an application you control. Partner Activities are somewhat in the middle.

As above, when returning data from Activities, you need to pay attention to information leakage from the destination application.

Example of returning data.

```
public void onReturnResultClick(View view) {

    // *** 4.1.1.1 - POINT 6 *** Information that is granted to be disclosed
    // to a partner application can be returned.
    Intent intent = new Intent();
    intent.putExtra("RESULT",
        "Information that is granted to disclose to partner applications");
    setResult(RESULT_OK, intent);
    finish();
}
```

4.1.2.8 Use the explicit Intents if the destination Activity is predetermined. (Required)

When using an Activity by implicit Intents, the Activity in which the Intent gets sent to is determined by the Android OS. If the Intent is mistakenly sent to malware then Information leakage can occur. On the other hand, when using an Activity by explicit Intents, only the intended Activity will receive the Intent so this is much safer.

Unless it is absolutely necessary for the user to determine which application's Activity the intent should be sent to, you should use explicit intents and specify the destination in advance.

Using an Activity in the same application by an explicit Intent

```
Intent intent = new Intent(this, PictureActivity.class);
intent.putExtra("BARCODE", barcode);
startActivity(intent);
```

Using other applicaion's Public Activity by an explicit Intent

```
Intent intent = new Intent();
intent.setClassName(
    "org.jssec.android.activity.publicactivity",
    "org.jssec.android.activity.publicactivity.PublicActivity");
startActivity(intent);
```

However, even when using another application's Public Activity by explicit Intents, it is possible that the destination Activity could be malware. This is because even if you limit the destination by package name, it is still possible that a malicious application can fake the same package name as the real application. To eliminate this type of risk, it is necessary to consider using a Partner or In-house.

Please refer to "4.1.3.1. *Combination of Exported Attribute and Intent Filter Setting (For Activity)*".

4.1.2.9 Handle the Returned Data from a Requested Activity Carefully and Securely (Required)

While the risks differ slightly according to what type of Activity you accessing, when processing Intent data received as a returned value, you always need to perform input validation on the received data.

Public Activities have to accept returned Intents from untrusted sources so when accessing a Public Activity it is possible that, the returned Intents are actually sent by malware. It is often mistakenly thought that all returned Intents from a Private Activity are safe because they are originating from the same application. However, since it is possible that an intent received from an untrusted source is indirectly forwarded, you should not blindly trust the contents of that Intent. Partner and In-house Activities have a risk somewhat in the middle of Private and Public Activities. Be sure to input validate these Activities as well.

Please refer to "3.2. *Handling Input Data Carefully and Securely*" for more information.

4.1.2.10 Verify the Destination Activity if Linking with Another Company's Application (Required)

Be sure to sure a whitelist when linking with another company's application. You can do this by saving a copy of the company's certificate hash inside your application and checking it with the certificate hash of the destination application. This will prevent a malicious application from being able to spoof Intents. Please refer to sample code section "4.1.1.3. *Creating/Using Partner Activities*" for the concrete implementation method. For technical details, please refer to "4.1.3.2. *Validating the Requesting Application*."

4.1.2.11 When Providing an Asset Secondhand, the Asset should be Protected with the Same Level of Protection (Required)

When an information or function asset, which is protected by a permission, is provided to another application secondhand, you need to make sure that it has the same required permissions needed to access the asset. In the Android OS permission security model, only an application that has been granted proper permissions can directly access a protected asset. However, there is a loophole because an application with permissions to an asset can act as a proxy and allow access to an unprivileged application. Substantially this is the same as re-delegating a permission so it is referred to as the "Permission Re-delegation" problem. Please refer to "5.2.3.4. *Permission Re-delegation Problem*."

4.1.2.12 Sending Sensitive Information Should Be Limited as much as possible (Recommended)

You should not send sensitive information to untrusted parties. Even when you are linking with a specific application, there is still a chance that you unintentionally send an Intent to a different application or that a malicious third party can steal your Intents. Please refer to "4.1.3.4. *Log Output When using Activities*."

You need to consider the risk of information leakage when sending sensitive information to an Activity. You must assume that all data in Intents sent to a Public Activity can be obtained by a malicious third party. In addition, there is a variety of risks of information leakage when sending Intents to Partner or In-house Activities as well depending on the implementation. Even when sending data to Private Activities, there is a risk that the data in the Intent could be leaked through LogCat. Information in the extras part of the Intent is not output to LogCat so it is best to store sensitive information there.

However, not sending sensitive data in the first place is the only perfect solution to prevent information leakage therefore you should limit the amount of sensitive information being sent as much as possible. When it is necessary to send sensitive information, the best practice is to only send to a trusted Activity and to make sure the information cannot be leaked through LogCat.

In addition, sensitive information should never be sent to the root Activity. Root Activities are Activities that are called first when a task is created. For example, the Activity which is launched from launcher is always the root Activity.

Please refer to "4.1.3.3. *About Root Activity*" for more details on root Activities.

4.1.3 Advanced Topics

4.1.3.1 Combination of Exported Attribute and Intent Filter Setting (For Activity)

We have explained how to implement the four types of Activities in this guidebook: Private Activities, Public Activities, Partner Activities, and In-house Activities. The various combinations of permitted settings for each type of exported attribute defined in the AndroidManifest.xml file and the intent-filter elements are defined in the table below. Please verify the compatibility of the exported attribute and intent-filter element with the Activity you are trying to create.

Table 4.1.2: Combination of exported setting and intent-filter setting

Value of exported attribute	intent-filter defined	intent-filter not defined
true	Public	Public, Partner-only, In-house only
false	(Prohibited)	Private
Unspecified	(Prohibited)	(Prohibited)

When the exported attribute of an Activity is left unspecified, the question of whether or not the Activity is public is determined by the presence or absence of Intent filters for that Activity[4]. However, in this guidebook it is forbidden to set the exported attribute to unspecified. In general, as mentioned previously, it is best to avoid implementations that rely on the default behavior of any given API; moreover, in cases where explicit methods exist for configuring important security-related settings such as the exported attribute, it is always a good idea to make use of those methods.

The reason why "an undefined Intent filter and an exported attribute of false" should not be used is that there is a loophole in Android's behavior, and because of how Intent filters work, other application's Activities can be called unexpectedly. The following two figures below show this explanation. Fig. 4.1.4 is an example of normal behavior in which a Private Activity (application A) can be called by an implicit Intent only from the same application. The Intent filter (action="X" in the figure) is defined only in application A, so this is the expected behavior.

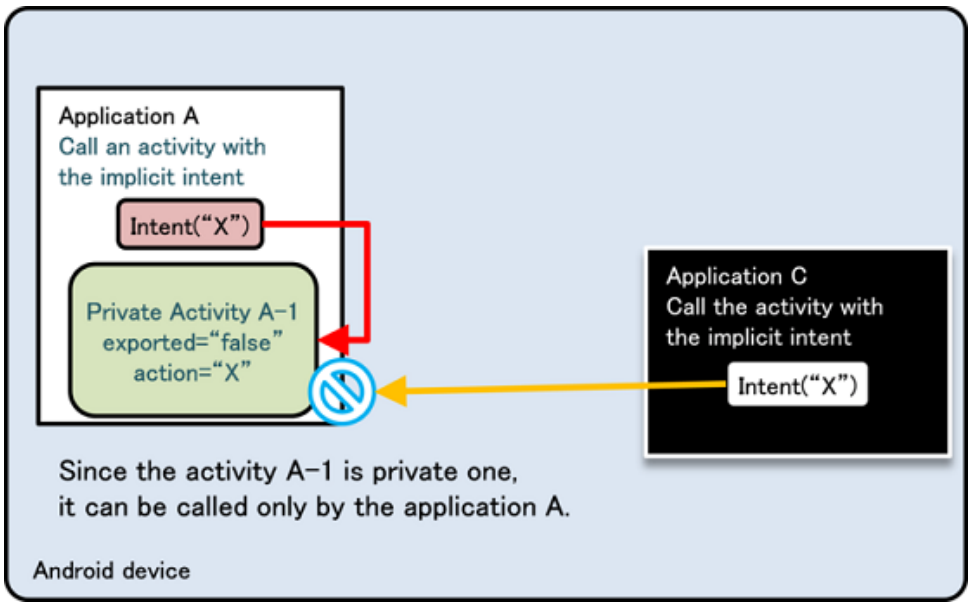


Fig. 4.1.4: An Example of Normal Behavior

Fig. 4.1.5 below shows a scenario in which the same Intent filter (action="X") is defined in Application B as well as Application A. In Fig. 4.1.5, Application A is trying to call a Private Activity in the same application by sending an implicit Intent, but this time, a dialog box asking the user which application to select is displayed, and the Public Activity B-1 in Application B is called by mistake due to the user selection[5]. Due to this loophole, it is possible that sensitive information can be sent to other applications or the application may receive an unexpected return value.

[4] If any Intent filters are defined, the Activity is public; otherwise it is private. Refer https://developer.android.com/guide/topics/manifest/activity-element.html#exported

[5] For terminals running Android 8.0 (API Level 26) or later, it has been confirmed that the "Complete action using" dialog is not displayed and an automatic transition is made to the Public Activity B-1 in Application B in the figure. For this reason, it should be prohibited to start a Private Activity with Intent filters by an implicit Intent.

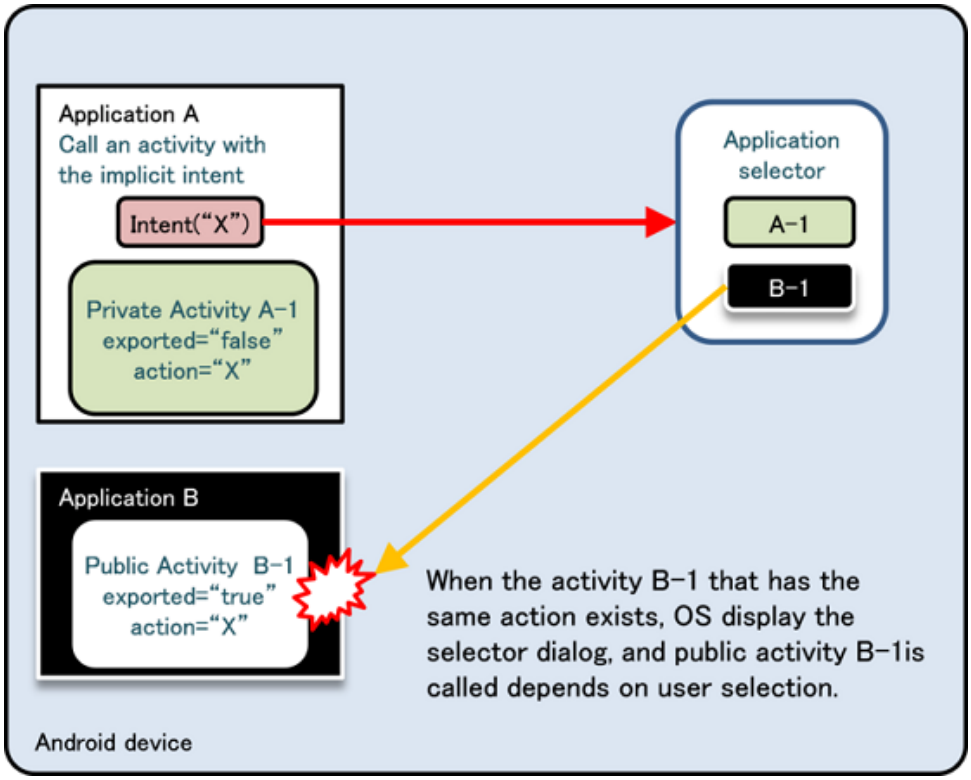


Fig. 4.1.5: An Example of Abnormal Behavior

As shown above, using Intent filters to send implicit Intents to Private Activities may result in unexpected behavior so it is best to avoid this setting. In addition, we have verified that this behavior does not depend on the installation order of Application A and Application B.

Note that from Android 12 onwards, the exported attribute is required. For details, please refer to 4.7.3.1. *Component Export Control and Intent Sending Restrictions*.

4.1.3.2 Validating the Requesting Application

Here we explain the technical information about how to implement a Partner Activity. Partner applications permit that only particular applications which are registered in a whitelist are allowed access and all other applications are denied. Because applications other than in-house applications also need access permission, we cannot use signature permissions for access control.

Simply speaking, we want to validate the application trying to use the Partner Activity by checking if it is registered in a predefined whitelist and allow access if it is and deny access if it is not. Application validation is done by obtaining the certificate from the application requesting access and comparing its hash with the one in the whitelist.

Some developers may think that it is sufficient to just compare "the package name" without obtaining "the certificate", however, it is easy to spoof the package name of a legitimate application so this is not a good method to check for authenticity. Arbitrarily assignable values should not be used for authentication. On the other hand, because only the application developer has the developer key for signing its certificate, this is a better method for identification. Since the certificate cannot be easily spoofed, unless a malicious third party can steal the developer key, there is a very small chance that malicious application will be trusted. While it is possible to store the entire certificate in the whitelist, it is sufficient to only store the SHA-256 hash value in order to minimize the file size.

There are two restrictions for using this method.

- The requesting application has to use startActivityForResult() instead of startActivity().
- The requesting application can only call from an Activity.

The second restriction is the restriction imposed as a result of the first restriction, so technically there is only a single restriction.

This restriction occurs due to the restriction of Activity.getCallingPackage() which gets the package name of the calling application. Activity.getCallingPackage() returns the package name of source (requesting) application only in case it is called by startActivityForResult(), but unfortunately, when it is called by startActivity(), it only returns null. Because of this, when using the method explained here, the source (requesting) application needs to use startActivityForResult() even if it does not need to obtain a return value. In addition, startActivityForResult() can be used only in Activity classes, so the source (requester) is limited to Activities.

PartnerActivity.java

```
/*
 * Copyright (C) 2012-2026 Japan Smartphone Security Association
 *
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 *
 *      http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */

package org.jssec.android.activity.partneractivity;

import org.jssec.android.shared.PkgCertWhitelists;
import org.jssec.android.shared.Utils;

import android.app.Activity;
import android.content.Context;
import android.content.Intent;
import android.os.Bundle;
import android.view.View;
import android.widget.Toast;

public class PartnerActivity extends Activity {

    // *** POINT 4 *** Verify the requesting application's certificate
    // through a predefined whitelist.
    private static PkgCertWhitelists sWhitelists = null;
    private static void buildWhitelists(Context context) {
        boolean isdebug = Utils.isDebuggable(context);
        sWhitelists = new PkgCertWhitelists();

        // Register certificate hash value of partner application
        // org.jssec.android.activity.partneruser.
        sWhitelists.add("org.jssec.android.activity.partneruser", isdebug ?
            // Certificate hash value of "androiddebugkey" in the debug.keystore.
            "0EFB7236 328348A9 89718BAD DF57F544 D5CCB4AE B9DB34BC 1E29DD26␣
↪F77C8255" :
            // Certificate hash value of "partner key" in the keystore.
            "1F039BB5 7861C27A 3916C778 8E78CE00 690B3974 3EB8259F E2627B8D␣
↪4C0EC35A");
```

(continues on next page)

(continued from previous page)

```
        // Register the other partner applications in the same way.
    }
    private static boolean checkPartner(Context context, String pkgname) {
        if (sWhitelists == null) buildWhitelists(context);
        return sWhitelists.test(context, pkgname);
    }

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);

        // *** POINT 4 *** Verify the requesting application's certificate
        // through a predefined whitelist.
        if (!checkPartner(this, getCallingActivity().getPackageName())) {
            Toast.makeText(this,
                           "Requesting application is not a partner application.",
                           Toast.LENGTH_LONG).show();
            finish();
            return;
        }

        // *** POINT 5 *** Handle the received intent carefully and securely,
        // even though the intent was sent from a partner application.
        // Omitted, since this is a sample. Refer to
        // "3.2 Handling Input Data Carefully and Securely."
        Toast.makeText(this, "Accessed by Partner App", Toast.LENGTH_LONG).show();
    }

    public void onReturnResultClick(View view) {

        // *** POINT 6 *** Only return Information that is granted to be disclosed
        // to a partner application.
        Intent intent = new Intent();
        intent.putExtra("RESULT", "Information for partner applications");
        setResult(RESULT_OK, intent);
        finish();
    }
}
```

PkgCertWhitelists.java

```
/*
 * Copyright (C) 2012-2026 Japan Smartphone Security Association
 *
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 *
 *      http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */
```

(continues on next page)

(continued from previous page)

```
package org.jssec.android.shared;

import android.content.pm.PackageManager;
import java.util.HashMap;
import java.util.Map;
import android.content.Context;
import android.os.Build;

import static android.content.pm.PackageManager.CERT_INPUT_SHA256;

public class PkgCertWhitelists {
    private Map<String, String> mWhitelists = new HashMap<String, String>();

    public boolean add(String pkgname, String sha256) {
        if (pkgname == null) return false;
        if (sha256 == null) return false;

        sha256 = sha256.replaceAll(" ", "");
        if (sha256.length() != 64)
            return false;  // SHA-256 -> 32 bytes -> 64 chars
        sha256 = sha256.toUpperCase();
        if (sha256.replaceAll("[0-9A-F]+", "").length() != 0)
            return false;  // found non hex char

        mWhitelists.put(pkgname, sha256);
        return true;
    }

    public boolean test(Context ctx, String pkgname) {
        // Get the correct hash value which corresponds to pkgname.
        String correctHash = mWhitelists.get(pkgname);

        // Compare the actual hash value of pkgname with the correct hash value.
        if (Build.VERSION.SDK_INT >= 28) {
            // ** if API Level >= 28, direct checking is possible
            PackageManager pm = ctx.getPackageManager();
            return pm.hasSigningCertificate(pkgname,
                                            Utils.hex2Bytes(correctHash),
                                            CERT_INPUT_SHA256);
        } else {
            // else use the facility of PkgCert
            return PkgCert.test(ctx, pkgname, correctHash);
        }
    }
}
```

PkgCert.java

```
/*
 * Copyright (C) 2012-2026 Japan Smartphone Security Association
 *
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 *
 *      http://www.apache.org/licenses/LICENSE-2.0
```

(continues on next page)

(continued from previous page)

```
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */

package org.jssec.android.shared;

import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;

import android.content.Context;
import android.content.pm.PackageInfo;
import android.content.pm.PackageManager;
import android.content.pm.PackageManager.NameNotFoundException;
import android.content.pm.Signature;

public class PkgCert {

    public static boolean test(Context ctx, String pkgname, String correctHash) {
        if (correctHash == null) return false;
        correctHash = correctHash.replaceAll(" ", "");
        return correctHash.equals(hash(ctx, pkgname));
    }

    public static String hash(Context ctx, String pkgname) {
        if (pkgname == null) return null;
        try {
            PackageManager pm = ctx.getPackageManager();
            PackageInfo pkginfo =
                pm.getPackageInfo(pkgname, PackageManager.GET_SIGNATURES);
            // Will not handle multiple signatures.
            if (pkginfo.signatures.length != 1) return null;
            Signature sig = pkginfo.signatures[0];
            byte[] cert = sig.toByteArray();
            byte[] sha256 = computeSha256(cert);
            return byte2hex(sha256);
        } catch (NameNotFoundException e) {
            return null;
        }
    }

    private static byte[] computeSha256(byte[] data) {
        try {
            return MessageDigest.getInstance("SHA-256").digest(data);
        } catch (NoSuchAlgorithmException e) {
            return null;
        }
    }

    private static String byte2hex(byte[] data) {
        if (data == null) return null;
        final StringBuilder hexadecimal = new StringBuilder();
        for (final byte b : data) {
```

(continues on next page)

(continued from previous page)

```
            hexadecimal.append(String.format("%02X", b));
        }
        return hexadecimal.toString();
    }
}
```

4.1.3.3 About Root Activity

ActivityActivityActivityActivityActivityActivity

Android 5.0API Level 21ActivityIntentActivityAndroid 7

- taskAffinity
- launchModestandard
- ActivityIntentFLAG_ACTIVITY_NEW_TASK

About Root Activity

The root Activity is the Activity that serves as the starting point of a task. It refers to the Activity that is first launched when the task is created. For example, if an Activity with default settings is launched from the launcher, that Activity becomes the root Activity.

Due to changes in Android 5.0 (API Level 21) and later, the risk of the Intent content sent to the root Activity being read by other apps has been resolved. Therefore, the previous security measure—such as "do not send sensitive information to the root Activity"—is no longer necessary from Android 7 onwards.

However, to prevent task management complexity and unexpected behavior, it is desirable to observe the following points in app design:

- Do not specify taskAffinity unless there is a special reason.
- Do not specify launchMode unless necessary (the default "standard" is recommended).
- Avoid casually setting FLAG_ACTIVITY_NEW_TASK on Intents sent to Activities.

About Activity launch modes

The launch mode of an Activity can be set by the android:launchMode attribute in the AndroidManifest.xml. If not specified, it is considered "standard." Also, the launch mode is affected by flags set in the Intent. For example, the FLAG_ACTIVITY_NEW_TASK flag causes the Activity to be launched in "singleTask" mode.

There are five types of launch modes for Activities:

Table 4.1.3: List of Activity Launch Modes

Launch Mode	Description
standard	The default mode; a new instance is created every time it is called. It belongs to the caller's task.
singleTop	Similar to standard, but if the same Activity is already at the top, no new instance is created; the existing instance is reused.
singleTask	If there is no existing task with a matching affinity, a new task is created with this Activity as the root. Otherwise, the existing task is reused.
singleInstance	Similar to singleTask, but only this Activity can belong to the created task. It is always the root Activity.
singleInstan-cePerTask	Available since Android 12; can only exist as a root Activity per task.

Activities launched with singleTask or singleInstance can be the root of a task. However, since Android 7 and later, there is no risk of the Intent content being read by other apps even when using these modes. Because complex task

structures may affect app behavior, it is recommended to use the standard mode as a basic rule unless for special use cases.

Since Android 7, the risk of Intents sent to the root Activity being read by other apps does not exist. Therefore, previous security measures are unnecessary, but it is still recommended for app design and user experience to avoid unnecessary specification of taskAffinity and launchMode, and casual use of FLAG_ACTIVITY_NEW_TASK.

4.1.3.4 Log Output When using Activities

When using an activity, the contents of intent are output to LogCat by ActivityManager. The following contents are to be output to LogCat, so in this case, sensitive information should not be included here.

- Destination Package name
- Destination Class name
- URI which is set by Intent#setData()

For example, when an application sent mails, the mail address is unfortunately outputted to LogCat if the application would specify the mail address to URI. So, better to send by setting Extras.

When sending a mail as below, mail address is shown to the logCat.

```
MainActivity.java
    // URI is output to the LogCat.
    Uri uri = Uri.parse("mailto:test@gmail.com");
    Intent intent = new Intent(Intent.ACTION_SENDTO, uri);
    startActivity(intent);
```

When using Extras, mail address is no more shown to the logCat.

```
MainActivity.java
    // Contents which was set to Extra, is not output to the LogCat.
    Uri uri = Uri.parse("mailto:");
    Intent intent = new Intent(Intent.ACTION_SENDTO, uri);
    intent.putExtra(Intent.EXTRA_EMAIL, new String[] {"test@gmail.com"});
    startActivity(intent);
```

However, there are cases where other applications can read the Extras data of intent using ActivityManager#getRecentTasks(). Please refer to "4.1.2.2. *Do Not Specify taskAffinity (Required)*", "4.1.2.3. *Do Not Specify launchMode (Required)*" and "4.1.2.4. *Do Not Set the FLAG_ACTIVITY_NEW_TASK Flag for Intents that Start an Activity (Required)*".

4.1.3.5 Protecting against Fragment Injection in the Settings Screen

The PreferenceActivity and the android.preference package have been deprecated since Android 10 (API Level 29). For future new development and maintenance, it is recommended to migrate to AndroidX's PreferenceFragmentCompat or similar.

When a class derived from PreferenceActivity is a public Activity, a problem known as *Fragment Injection*[6] may arise. However, in the current recommended implementation, this risk can be avoided by using PreferenceFragmentCompat.

When using PreferenceFragmentCompat, the risk of Fragment Injection basically does not exist. The settings screen is constructed with AppCompatActivity + PreferenceFragmentCompat.

```
class SettingsActivity : AppCompatActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
```

(continues on next page)

[6] For more information on Fragment Injection, consult this URL: https://securityintelligence.com/new-vulnerability-android-framework-fragment-injection/

(continued from previous page)

```
        setContentView(R.layout.activity_settings)

        if (savedInstanceState == null) {
            supportFragmentManager.beginTransaction()
                .replace(R.id.fragment_container, SettingsFragment())
                .commit()
        }
    }

    class SettingsFragment : PreferenceFragmentCompat() {
        override fun onCreatePreferences(savedInstanceState: Bundle?, rootKey:␣
↪String?) {
            setPreferencesFromResource(R.xml.preferences, rootKey)
        }
    }
}
```

If you have no choice but to continue using PreferenceActivity (e.g., for legacy device support), countermeasures against the Fragment Injection vulnerability are mandatory. You must override isValidFragment() and explicitly specify only the allowed Fragments. Furthermore, set android:exported="false" in the manifest to prevent unnecessary calls from the outside.

```
override fun isValidFragment(fragmentName: String): Boolean {
    return SettingsFragment::class.java.name == fragmentName
}
```

When obfuscating, it is necessary to set the -keepnames option with ProGuard or similar tools to prevent the class names from changing. Note that Google Play may block new releases or updates of apps containing PreferenceActivity that still have the Fragment Injection vulnerability. From the perspectives of maintainability and security, it is strongly recommended to avoid the use of PreferenceActivity and migrate to the AndroidX Preference library as soon as possible.

4.1.3.6 The Autofill framework

The Autofill framework was added in Android 8.0 (API Level 26). Using this framework allows apps to store information entered by users—such as user names, passwords, addresses, phone numbers, and credit cards—and subsequently to retrieve this information as necessary to allow the app to fill in forms automatically. This is a convenient mechanism that reduces data-entry burdens for users; however, because it allows a given app to pass sensitive information such as passwords and credit cards to other apps, it must be handled with appropriate care.

Overview of the framework

2 components

In what follows, we provide an overview of the two components[7] registered by the Autofill framework.

- Apps eligible for Autofill (user apps):
  - Pass view information (text and attributes) to Autofill service; receive information from Autofill service as needed to auto-fill forms.
  - All apps that have Activities are user apps (when in the foreground).
  - It is possible for all Views of all user apps to be eligible for Autofill. It is also possible to explicitly specify that any given individual view should be ineligible for Autofill.
  - It is also possible to restrict an app's use of Autofill to the Autofill service within the same package.
- Services that provide Autofill (Autofill services):

[7] The "user app" and the "Autofill service" may belong to the same package (the same APK file) or to different packages.

- Save View information passed by an app (requires user permission); provide an app with information needed for Autofill in a View (candidate lists).
- The Views eligible for this information saving are determined by the Autofill service. (Within the Autofill framework, by default information on all Views contained in an Activity are passed to the Autofill service.)
- It is also possible to construct Autofill services provided by third parties.
- It is possible for several to be present within a single terminal with only the service selected by the user via "Settings" enabled ("None" is also a possible selection.)
- It also possible for a Service to provide a UI to validate users via password entry or other mechanisms to protect the security of the user information handled.

Procedural flowchart for the Autofill framework

Fig. 4.1.6 is a flowchart illustrating the procedural flow of interactions among Autofill-related components during Autofill. When triggered by events such as motion of the focus in a user app's View, information on that View (primarily the parent-child relationships and various attributes of the View) is passed via the Autofill framework to the Autofill service selected within "Settings". Based on the data it receives, the Autofill service fetches from a database the information (candidate lists) needed for Autofill, then returns this to the framework. The framework displays a candidate list to the user, and the app carries out the Autofill operation using the data selected by the user.

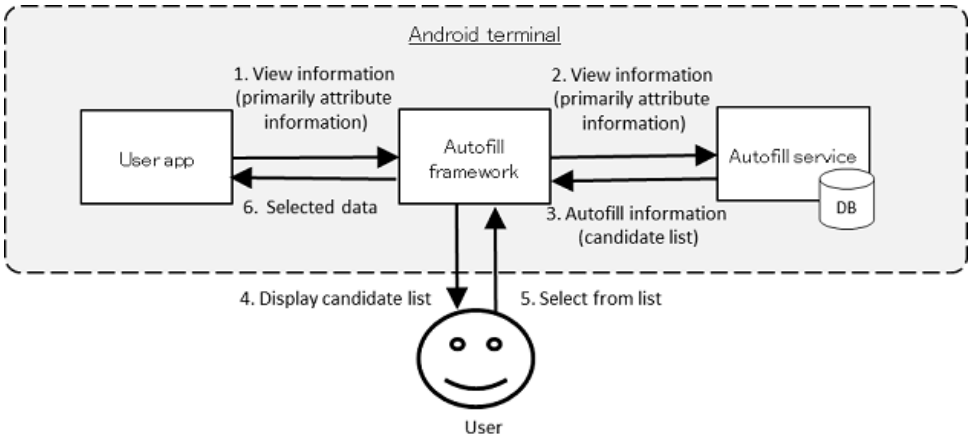


Fig. 4.1.6: Procedural flow among components for Autofill

Next, Fig. 4.1.7 is a flowchart illustrating the procedural flow for saving user data via Autofill. Upon a triggering event such as when AutofillManager#commit() is called or when an Activity is unfocused, if any Autofilled values for the View have been modified and the user has granted permission via the Save Permission dialog box displayed by the Autofill framework, information on the View (including text) is passed via the Autofill framework to the Autofill service selected via "Settings", and the Autofill service stores information in the database to complete the procedural sequence.

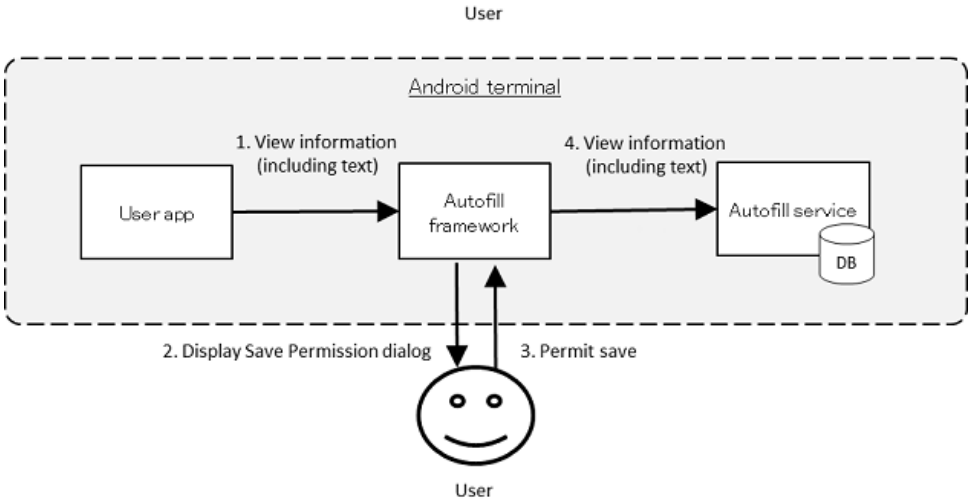


Fig. 4.1.7: Procedural flow among components for saving user data

Security concerns for Autofill user apps

As noted in the section "Overview of the framework" above, the security model adopted by the Autofill framework is premised on the assumption that the user configures the "Settings" to select secure Autofill services and makes appropriate decisions regarding which data to pass to which Autofill service when storing data.

However, if a user unwittingly selects a non-secure Autofill service, there is a possibility that the user may permit the storage of sensitive information that should not be passed to the Autofill service. In what follows we discuss the damage that could result in such a scenario.

When saving information, if the user selects an Autofill service and grants it permission via the Save Permission dialog box, information for all Views contained in the Activity currently displayed by the app in use may be passed to the Autofill service. If the Autofill service is malware, or if other security issues arise—for example, if View information is stored by the Autofill service on an external storage medium or on an insecure cloud service—this could create the risk that information handled by the app might be leaked.

On the other hand, during Autofill, if the user has selected a piece of malware as the Autofill service, values transmitted by the malware may be entered as input. At this point, if the security of the data input is not adequately validated by the app or by the cloud services to which the app sends data, risks of information leakage and/or termination of the app or the service may arise.

Note that, as discussed above in the section "2 components", apps with Activities are automatically eligible for Autofill, and thus all developers of apps with Activities must take the risks described above into account when designing and implementing apps. In what follows we will present countermeasures to mitigate the risks described above we recommend that these be adopted as appropriate based on a consideration of the countermeasures required by an app—referring to "3.1.3. *Asset Classification and Protective Countermeasures*" and other relevant resources.

Steps to mitigate risk: 1

As discussed above, security within the Autofill framework is ultimately guaranteed only at the user's discretion. For this reason, the range of countermeasures available to apps is somewhat limited. However, there is one way to mitigate the concerns described above: Setting the importantForAutofill attribute for a view to "no" ensures that no View information is passed to the Autofill service (i.e. the View is made ineligible for Autofill), even if the user cannot make appropriate selections or permissions (such as selecting a piece of malware as the Autofill service)[8].

The importantForAutofill attribute may be specified by any of the following methods.

- Set the importantForAutofill attribute in the layout XML
- Call View#setImportantForAutofill()

[8] Even after taking this step, in some cases it may not be possible to avoid the security concerns described above—for example, if the user intentionally uses Autofill. Implementing the steps described in "*Steps to mitigate risk*" will improve security in these cases.

The values that may be set for this attribute are shown below. Make sure to use values appropriate for the specified range. In particular, note with caution that, when a value is set to “no” for a View, that View will be ineligible for Autofill, but its children will remain eligible for Autofill. The default value is “auto.”

Table 4.1.4: Eligible for Autofill?

Value Name of constant	Specified View	Child View
“auto” IMPORTANT_FOR_AUTOFILL_AUTO	“auto”[9]	“auto”[9]
“no” IMPORTANT_FOR_AUTOFILL_NO	No	Yes
“noExcludeDescendants” IMPORTANT_FOR_AUTOFILL_NO_EXCLUDE_DESCENDANTS	No	No
“yes” IMPORTANT_FOR_AUTOFILL_YES	Yes	Yes
“yesExcludeDescendants” IMPORTANT_FOR_AUTOFILL_YES_EXCLUDE_DESCENDANTS	Yes	No

It is also possible to use AutofillManager#hasEnabledAutofillServices() to restrict the use of Autofill functionality to Autofill services within the same package.

In what follows, we show an example that all Views in an Activity are eligible for Autofill (whether or not a View actually uses Autofill is determined by the Autofill service) only in case that "Settings" have been configured to use a Autofill service within the same package. It is also possible to call View#setImportantForAutofill() for individual Views.

DisableForOtherServiceActivity.java

```
/*
 * Copyright (C) 2012-2026 Japan Smartphone Security Association
 *
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 *
 *      http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */
package org.jssec.android.autofillframework.autofillapp;

import android.os.Bundle;
import androidx.appcompat.app.AppCompatActivity;
import android.view.View;
import android.view.autofill.AutofillManager;
import android.widget.EditText;
import android.widget.TextView;

import org.jssec.android.autofillframework.R;
```

(continues on next page)

[9] Determined by Autofill framework

(continued from previous page)

```
public class DisableForOtherServiceActivity extends AppCompatActivity {
    private boolean mIsAutofillEnabled = false;

    private EditText mUsernameEditText;
    private EditText mPasswordEditText;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.disable_for_other_service_activity);

        mUsernameEditText = (EditText)findViewById(R.id.field_username);
        mPasswordEditText = (EditText)findViewById(R.id.field_password);

        findViewById(R.id.button_login).setOnClickListener(
            new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                login();
            }
        });

        findViewById(R.id.button_clear).setOnClickListener(
            new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                resetFields();
            }
        });

        //Because the floating-toolbar is not supported for this Activity,
        // Autofill may be used by selecting "Automatic Input"
    }

    @Override
    protected void onStart() {
        super.onStart();
    }

    @Override
    protected void onResume() {
        super.onResume();
        updateAutofillStatus();
        View rootView = this.getWindow().getDecorView();
        if (!mIsAutofillEnabled) {
            //If not using Autofill service within the same package,
            // make all Views ineligible for Autofill
            rootView.setImportantForAutofill(View.IMPORTANT_FOR_AUTOFILL_NO_
↪EXCLUDE_DESCENDANTS);
        } else {
            //If using Autofill service within the same package,
            // make all Views eligible for Autofill
            //View#setImportantForAutofill() may also be called for specific Views
            rootView.setImportantForAutofill(View.IMPORTANT_FOR_AUTOFILL_AUTO);
        }
    }
```

(continues on next page)

(continued from previous page)

```
    private void login() {
        String username = mUsernameEditText.getText().toString();
        String password = mPasswordEditText.getText().toString();

        //Validate data obtained from View
        if (!Util.validateUsername(username) || !Util.validatePassword(password)) {
            //appropriate error handling
        }

        //Send username, password to server

        finish();
    }

    private void resetFields() {
        mUsernameEditText.setText("");
        mPasswordEditText.setText("");
    }

    private void updateAutofillStatus() {
        AutofillManager mgr = getSystemService(AutofillManager.class);

        mIsAutofillEnabled = mgr.hasEnabledAutofillServices();

        TextView statusView = (TextView) findViewById(R.id.label_autofill_status);
        String status = "Our autofill service is --.";
        if (mIsAutofillEnabled) {
            status = "autofill service within same package is enabled";
        } else {
            status = "autofill service within same package is disabled";
        }
        statusView.setText(status);
    }
}
```

Steps to mitigate risk: 2

Even in cases where an app has implemented the steps described in the previous section (”*Steps to mitigate risk: 1*”), the user can forcibly enable the use of Autofill by long-pressing the View, displaying the floating toolbar or a similar control interface, and selecting "Automatic input". In this case, information for all Views—including Views for which the importantForAutofill attribute has been set to “no,” or for which similar steps have been taken—will be passed to the Autofill service.

It is possible to avoid the risk of information leakage even in circumstances such as these by deleting the “Automatic Input” option from the floating-toolbar menu and other control interfaces; this step is to be carried out in addition to the procedures described in “*Steps to mitigate risk: 1*”.

Sample code for this purpose is shown below.

DisableAutofillActivity.java

```
/*
 * Copyright (C) 2012-2026 Japan Smartphone Security Association
 *
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 *
```

(continues on next page)

(continued from previous page)

```
 *      http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */
package org.jssec.android.autofillframework.autofillapp;

import android.os.Bundle;
import androidx.appcompat.app.AppCompatActivity;
import android.view.ActionMode;
import android.view.Menu;
import android.view.MenuItem;
import android.view.SubMenu;
import android.view.View;
import android.widget.EditText;

import org.jssec.android.autofillframework.R;

public class DisableAutofillActivity extends AppCompatActivity {

    private EditText mUsernameEditText;
    private EditText mPasswordEditText;

    private ActionMode.Callback mActionModeCallback;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.disable_autofill_activity);

        mUsernameEditText = (EditText) findViewById(R.id.field_username);
        mPasswordEditText = (EditText) findViewById(R.id.field_password);

        findViewById(R.id.button_login).setOnClickListener(
            new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                login();
            }
        });

        findViewById(R.id.button_clear).setOnClickListener(
            new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                resetFields();
            }
        });

        mActionModeCallback = new ActionMode.Callback() {
            @Override
            public boolean onCreateActionMode(ActionMode mode, Menu menu) {
                removeAutofillFromMenu(menu);
```

(continues on next page)

(continued from previous page)

```
                return true;
            }

            @Override
            public boolean onPrepareActionMode(ActionMode mode, Menu menu) {
                removeAutofillFromMenu(menu);
                return true;
            }

            @Override
            public boolean onActionItemClicked(ActionMode mode, MenuItem item) {
                return false;
            }

            @Override
            public void onDestroyActionMode(ActionMode mode) {
            }
        };

        //Delete "Automatic Input" from floating-toolbar
        setMenu();
    }

    void setMenu() {
        if (mActionModeCallback == null) {
            return;
        }
        //Register callback for all editable TextViews contained in Activity
        mUsernameEditText
                .setCustomInsertionActionModeCallback(mActionModeCallback);
        mPasswordEditText
                .setCustomInsertionActionModeCallback(mActionModeCallback);
    }

    //Traverse all menu levels, deleting "Automatic Input" from each
    void removeAutofillFromMenu(Menu menu) {
        if (menu.findItem(android.R.id.autofill) != null) {
            menu.removeItem(android.R.id.autofill);
        }

        for (int i=0; i<menu.size(); i++) {
            SubMenu submenu = menu.getItem(i).getSubMenu();
            if (submenu != null) {
                removeAutofillFromMenu(submenu);
            }
        }
    }

    private void login() {
        String username = mUsernameEditText.getText().toString();
        String password = mPasswordEditText.getText().toString();

        //Validate data obtained from View
        if (!Util.validateUsername(username) || Util.validatePassword(password)) {
            //appropriate error handling
        }
```

(continues on next page)

(continued from previous page)

```
        //Send username, password to server

        finish();
    }

    private void resetFields() {
        mUsernameEditText.setText("");
        mPasswordEditText.setText("");
    }
}
```

Steps to mitigate risk: 3

In Android 9.0 (API level 28), AutofillManager#getAutofillServiceComponentName() can be used to find out what components of Autofill Service are currently enabled. This can be used to obtain the package name and confirm whether the application itself is considered a trusted Autofill Service.

In this case, as described in "4.1.3.2. *Validating the Requesting Application*" above, because a package name could be spoofed, identity verification solely using this method cannot be recommended. In the same way as the example described in 4.1.3.2., the Autofill Service certificate must be obtained from the package name, and the identity must be verified by checking that the certificate matches one that was registered beforehand in a whitelist. This method is described in detail in 4.1.3.2., and so refer to this section for more information.

An example is shown below where Autofill is used for all views of an activity only when an Autofill Service that was registered beforehand in the whitelist is enabled.

EnableOnlyWhitelistedServiceActivity.java

```
/*
 * Copyright (C) 2012-2026 Japan Smartphone Security Association
 *
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 *
 *      http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */

package org.jssec.android.autofillframework.autofillapp;

import android.content.ComponentName;
import android.content.Context;
import android.os.Bundle;
import android.app.Activity;
import android.view.View;
import android.view.autofill.AutofillManager;
import android.widget.EditText;
import android.widget.TextView;
import android.widget.Toast;

import org.jssec.android.shared.PkgCertWhitelists;
import org.jssec.android.autofillframework.R;
```

(continues on next page)

(continued from previous page)

```
public class EnableOnlyWhitelistedServiceActivity extends Activity {
    private static PkgCertWhitelists sWhitelists = null;
    private static void buildWhitelists(Context context) {
        sWhitelists = new PkgCertWhitelists();
        // Register hash value of the certificate of trusted Autofill Service
        sWhitelists.add("com.google.android.gms",
            "1975B2F17177BC89A5DFF31F9E64A6CAE281A53DC1D1D59B1D147FE1C82AFA00");
        // In a similer manner register other trusting Autofill Srvices
        //    :
    }
    private static boolean checkService(Context context, String pkgname) {
        if (sWhitelists == null) buildWhitelists(context);
        return sWhitelists.test(context, pkgname);
    }

    private boolean mIsAutofillEnabled = false;

    private EditText mUsernameEditText;
    private EditText mPasswordEditText;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.enable_only_whitelisted_service_activity);

        mUsernameEditText = (EditText)findViewById(R.id.field_username);
        mPasswordEditText = (EditText)findViewById(R.id.field_password);

        findViewById(R.id.button_login).setOnClickListener(
            new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                login();
            }
        });

        findViewById(R.id.button_clear).setOnClickListener(
            new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                resetFields();
            }
        });
        // Because the floating-toolbar is not supported for this Activity,
        // Autofill may be used by selecting "Automatic Input"
    }

    @Override
    protected void onStart() {
        super.onStart();
    }

    @Override
    protected void onResume() {
        super.onResume();
```

(continues on next page)

(continued from previous page)

```
        updateAutofillStatus();
        View rootView = this.getWindow().getDecorView();
        if (!mIsAutofillEnabled) {
            // If the Autofill Service is not on white list,
            // exclude all Views from the target of Autofill
            rootView.setImportantForAutofill(View.IMPORTANT_FOR_AUTOFILL_NO_
↪EXCLUDE_DESCENDANTS);
        } else {
            // If the Autofill Service is on white list,
            // include all Views as the target of Autofill
            // It is also possible to call View#setImportantForAutofill()
            // for a specific View
            rootView.setImportantForAutofill(View.IMPORTANT_FOR_AUTOFILL_AUTO);
        }
    }
    private void login() {
        String username = mUsernameEditText.getText().toString();
        String password = mPasswordEditText.getText().toString();

        // Validate safetiness of data obtained from View
        if (!Util.validateUsername(username) || !Util.validatePassword(password)) {
            // Do apropriate error handling
        }

        // Eend username and passowrd to the Server

        finish();
    }

    private void resetFields() {
        mUsernameEditText.setText("");
        mPasswordEditText.setText("");
    }

    private void updateAutofillStatus() {
        AutofillManager mgr = getSystemService(AutofillManager.class);
        // From Android 9.0 (API Level 28), it is possible to get
        // component info. of Autofill Service
        ComponentName componentName = mgr.getAutofillServiceComponentName();
        String componentNameString = "None";
        if (componentName == null) {
            // "Settings"-"Autofill Service" is set to "None"
            mIsAutofillEnabled = false;
            Toast.makeText(this, "No Autofill Service", Toast.LENGTH_LONG).show();
        } else {
            String autofillServicePackage = componentName.getPackageName();
            // Check if the Autofill Service is registered in white list
            if (checkService(this, autofillServicePackage)) {
                mIsAutofillEnabled = true;
                Toast.makeText(this,
                    "Trusted Autofill Service: " + autofillServicePackage,
                    Toast.LENGTH_LONG).show();
            } else {
                Toast.makeText(this,
                    "Untrusted Autofill Service: " + autofillServicePackage,
                    Toast.LENGTH_LONG).show();
```

(continues on next page)

(continued from previous page)

```
                // if not on white list, do not use Autofill Service
                mIsAutofillEnabled = false;
            }
            componentNameString =
                    autofillServicePackage + " / " + componentName.getClassName();
        }

        TextView statusView = (TextView) findViewById(R.id.label_autofill_status);
        String status = "current autofill service: \n" + componentNameString;
        statusView.setText(status);
    }
}
```

PkgCertWhitelists.java

```
/*
 * Copyright (C) 2012-2026 Japan Smartphone Security Association
 *
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 *
 *      http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */

package org.jssec.android.shared;

import android.content.pm.PackageManager;
import java.util.HashMap;
import java.util.Map;
import android.content.Context;
import android.os.Build;

import static android.content.pm.PackageManager.CERT_INPUT_SHA256;

public class PkgCertWhitelists {
    private Map<String, String> mWhitelists = new HashMap<String, String>();

    public boolean add(String pkgname, String sha256) {
        if (pkgname == null) return false;
        if (sha256 == null) return false;

        sha256 = sha256.replaceAll(" ", "");
        if (sha256.length() != 64)
            return false;  // SHA-256 -> 32 bytes -> 64 chars
        sha256 = sha256.toUpperCase();
        if (sha256.replaceAll("[0-9A-F]+", "").length() != 0)
            return false;  // found non hex char

        mWhitelists.put(pkgname, sha256);
        return true;
```

(continues on next page)

(continued from previous page)

```
    }

    public boolean test(Context ctx, String pkgname) {
        // Get the correct hash value which corresponds to pkgname.
        String correctHash = mWhitelists.get(pkgname);

        // Compare the actual hash value of pkgname with the correct hash value.
        if (Build.VERSION.SDK_INT >= 28) {
            // ** if API Level >= 28, direct checking is possible
            PackageManager pm = ctx.getPackageManager();
            return pm.hasSigningCertificate(pkgname,
                                            Utils.hex2Bytes(correctHash),
                                            CERT_INPUT_SHA256);
        } else {
            // else use the facility of PkgCert
            return PkgCert.test(ctx, pkgname, correctHash);
        }
    }
}
```

PkgCert.java

```
/*
 * Copyright (C) 2012-2026 Japan Smartphone Security Association
 *
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 *
 *      http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */

package org.jssec.android.shared;

import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;

import android.content.Context;
import android.content.pm.PackageInfo;
import android.content.pm.PackageManager;
import android.content.pm.PackageManager.NameNotFoundException;
import android.content.pm.Signature;

public class PkgCert {

    public static boolean test(Context ctx, String pkgname, String correctHash) {
        if (correctHash == null) return false;
        correctHash = correctHash.replaceAll(" ", "");
        return correctHash.equals(hash(ctx, pkgname));
    }
```

(continues on next page)

(continued from previous page)

```
    public static String hash(Context ctx, String pkgname) {
        if (pkgname == null) return null;
        try {
            PackageManager pm = ctx.getPackageManager();
            PackageInfo pkginfo =
                pm.getPackageInfo(pkgname, PackageManager.GET_SIGNATURES);
            // Will not handle multiple signatures.
            if (pkginfo.signatures.length != 1) return null;
            Signature sig = pkginfo.signatures[0];
            byte[] cert = sig.toByteArray();
            byte[] sha256 = computeSha256(cert);
            return byte2hex(sha256);
        } catch (NameNotFoundException e) {
            return null;
        }
    }

    private static byte[] computeSha256(byte[] data) {
        try {
            return MessageDigest.getInstance("SHA-256").digest(data);
        } catch (NoSuchAlgorithmException e) {
            return null;
        }
    }

    private static String byte2hex(byte[] data) {
        if (data == null) return null;
        final StringBuilder hexadecimal = new StringBuilder();
        for (final byte b : data) {
            hexadecimal.append(String.format("%02X", b));
        }
        return hexadecimal.toString();
    }
}
```

4.1.3.7 Strengthening Restrictions on Background Activity Launch (BAL)

In Android 17, restrictions on Background Activity Launch (BAL) have been further strengthened to prevent unintended, unauthorized screen transitions by users and abuse by malware. Even in traditional Android, launching a screen (Activity) directly from a background state by an app was restricted under certain conditions, but in Android 17, those conditions have become stricter, being reinforced in a direction where screen launching without explicit user operation is, as a rule, not possible. In particular, behaviors that forcibly display a screen at unnatural timing even via notifications or foreground services are subject to restrictions, designed to reduce the risks of impeding user experience and fraudulent guidance such as phishing.

Behind this change are countermeasures against attack techniques that steal credentials by suddenly displaying login screens or fake input screens from the background, as well as nuisance behaviors that forcibly display advertisements or unnecessary screens. In Android 17, by suppressing such behaviors at the OS level, progress is being made toward placing importance on safe and predictable user operations. Therefore, developers need to transition to a design where, rather than launching an Activity directly from the background as before, they prompt user action via notifications (Notification) and display screens triggered by explicit operations such as taps. In this way, strengthening BAL restrictions serves as an important change to protect both security and UX (user experience).

MODE_BACKGROUND_ACTIVITY_START_ALLOW_IF_VISIBLE is one of the controls for Background Activity Launch (BAL) in Android, and is a configuration for permitting screen launches from the background only when the app is in a state visible to the user. In Android, an app in the background suddenly displaying a screen without user operation leads to phishing, unauthorized screen guidance, and degradation of user experience, so it is strictly

restricted by default. When this mode is specified, if the app is in a state recognizable by the user, such as operating in the foreground or being even partially displayed on the screen, it becomes possible to launch a new Activity via PendingIntent or similar. On the other hand, when the app is completely in the background and invisible to the user, launching is not permitted, and user operation via notifications is required as before. By making visibility a condition in this way, the design maintains security while respecting the user's current operation context.

The following shows an implementation example of sending a PendingIntent specifying MODE_BACKGROUND_ACTIVITY_START_ALLOW_IF_VISIBLE introduced in Android 17.

```
val intent = Intent(context, TargetActivity::class.java)
val pendingIntent = PendingIntent.getActivity(
    context,
    0,
    intent,
    PendingIntent.FLAG_UPDATE_CURRENT or PendingIntent.FLAG_IMMUTABLE
)

// Create ActivityOptions
val options = ActivityOptions.makeBasic()

// Allow launch from the background only when the caller is visible
options.pendingIntentBackgroundActivityStartMode =
    ActivityOptions.MODE_BACKGROUND_ACTIVITY_START_ALLOW_IF_VISIBLE

try {
    // Convert options to Bundle and send PendingIntent
    pendingIntent.send(
        context,
        0,
        intent,
        null,
        null,
        null,
        options.toBundle()
    )
} catch (e: PendingIntent.CanceledException) {
    e.printStackTrace()
}
```

4.2 Receiving/Sending Broadcasts

4.2.1 Sample Code

Creating Broadcast Receiver is required to receive Broadcast. Risks and countermeasures of using Broadcast Receiver differ depending on the type of the received Broadcast.

You can find your Broadcast Receiver in the following judgment flow. The receiving applications cannot check the package names of Broadcast-sending applications that are necessary for linking with the partners. As a result, Broadcast Receiver for the partners cannot be created.

Table 4.2.1: Definition of broadcast receiver types

Type	Definition
Private broadcast receiver	A broadcast receiver that can receive broadcasts only from the same application, therefore is the safest broadcast receiver
Public broadcast receiver	A broadcast receiver that can receive broadcasts from an unspecified large number of applications.
In-house broadcast receiver	A broadcast receiver that can receive broadcasts only from other In-house applications

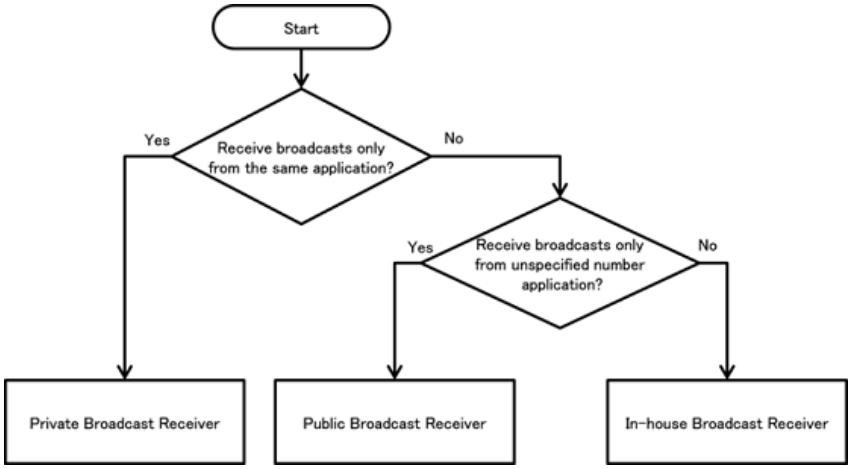


Fig. 4.2.1: Flow Figure to select Broadcast Receiver Type

In addition, Broadcast Receiver can be divided into 2 types based on the definition methods, Static Broadcast Receiver and Dynamic Broadcast Receiver. The differences between them can be found in the following figure. In the sample code, an implementation method for each type is shown. The implementation method for sending applications is also described because the countermeasure for sending information is determined depending on the receivers.

Table 4.2.2: Definition Method and Characteristic of Broadcast Receivers

	Definition method	Characteristic
Static Broadcast Receiver	Define by writing <receiver> elements in AndroidManifest.xml	• There is a restriction that some Broadcasts (e.g. ACTION_BATTERY_CHANGED) sent by system cannot be received. • Broadcast can be received from application's initial boot till uninstallation. • If the app's targetSDKVersion is 26 or above, then, on terminals running Android 8.0 (API level 26) or later, Broadcast Receivers may not be registered for implicit Broadcast Intents[10]
Dynamic Broadcast Receiver	By calling registerReceiver() and unregisterReceiver() in a program, register/unregister Broadcast Receiver dynamically.	• Broadcasts which cannot be received by static Broadcast Receiver can be received. • The period of receiving Broadcasts can be controlled by the program. For example, Broadcasts can be received only while Activity is on the front side. • Private Broadcast Receiver cannot be created.

[10] As exceptions to this rule, some implicit Broadcast Intents sent by the system may use Broadcast Receivers. For more information, consult the following URL. https://developer.android.com/guide/components/broadcast-exceptions.html

4.2.1.1 Private Broadcast Receiver - Receiving/Sending Broadcasts

Private Broadcast Receiver is the safest Broadcast Receiver because only Broadcasts sent from within the application can be received. Dynamic Broadcast Receiver cannot be registered as Private, so Private Broadcast Receiver consists of only Static Broadcast Receivers.

Points (Receiving Broadcasts):

1. Explicitly set the exported attribute to false.
2. Handle the received intent carefully and securely, even though the intent was sent from within the same application.
3. Sensitive information can be sent as the returned results since the requests come from within the same application.

AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
          >

  <application
      android:icon="@drawable/ic_launcher"
      android:label="@string/app_name"
      android:allowBackup="false" >

    <!-- Private Broadcast Receiver -->
    <!-- *** POINT 1 *** Explicitly set the exported attribute to false. -->
    <receiver
        android:name=".PrivateReceiver"
        android:exported="false" />

    <activity
        android:name=".PrivateSenderActivity"
        android:label="@string/app_name"
        android:exported="true" >
      <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER" />
      </intent-filter>
    </activity>
  </application>
</manifest>
```

PrivateReceiver.java

```
/*
 * Copyright (C) 2012-2026 Japan Smartphone Security Association
 *
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 *
 *      http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
```

(continues on next page)

(continued from previous page)

```
 */

package org.jssec.android.broadcast.privatereceiver;

import android.app.Activity;
import android.content.BroadcastReceiver;
import android.content.Context;
import android.content.Intent;
import android.widget.Toast;

public class PrivateReceiver extends BroadcastReceiver {

    @Override
    public void onReceive(Context context, Intent intent) {

        // *** POINT 2 *** Handle the received intent carefully and securely,
        // even though the intent was sent from within the same application.
        // Omitted, since this is a sample. Please refer to
        // "3.2 Handling Input Data Carefully and Securely."
        String param = intent.getStringExtra("PARAM");
        Toast.makeText(context,
                       String.format("Received param: \"%s\"", param),
                       Toast.LENGTH_SHORT).show();

        // *** POINT 3 *** Sensitive information can be sent as the returned
        // results since the requests come from within the same application.
        setResultCode(Activity.RESULT_OK);
        setResultData("Sensitive Info from Receiver");
        abortBroadcast();
    }
}
```

The sample code for sending Broadcasts to private Broadcast Receiver is shown below.

Points (Sending Broadcasts):

4. Use the explicit Intent with class specified to call a receiver within the same application.
5. Sensitive information can be sent since the destination Receiver is within the same application.
6. Handle the received result data carefully and securely, even though the data came from the Receiver within the same application.

PrivateSenderActivity.java

```
/*
 * Copyright (C) 2012-2026 Japan Smartphone Security Association
 *
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 *
 *      http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
```

(continues on next page)

(continued from previous page)

```
 */

package org.jssec.android.broadcast.privatereceiver;

import android.app.Activity;
import android.content.BroadcastReceiver;
import android.content.Context;
import android.content.Intent;
import android.os.Bundle;
import android.view.View;
import android.widget.TextView;

public class PrivateSenderActivity extends Activity {

    public void onSendNormalClick(View view) {
        // *** POINT 4 *** Use the explicit Intent with class specified to call
        // a receiver within the same application.
        Intent intent = new Intent(this, PrivateReceiver.class);

        // *** POINT 5 *** Sensitive information can be sent since the destination
        // Receiver is within the same application.
        intent.putExtra("PARAM", "Sensitive Info from Sender");
        sendBroadcast(intent);
    }

    public void onSendOrderedClick(View view) {
        // *** POINT 4 *** Use the explicit Intent with class specified to call
        // a receiver within the same application.
        Intent intent = new Intent(this, PrivateReceiver.class);

        // *** POINT 5 *** Sensitive information can be sent since the destination
        // Receiver is within the same application.
        intent.putExtra("PARAM", "Sensitive Info from Sender");
        sendOrderedBroadcast(intent, null, mResultReceiver, null, 0, null, null);
    }

    private BroadcastReceiver mResultReceiver = new BroadcastReceiver() {
            @Override
            public void onReceive(Context context, Intent intent) {

                // *** POINT 6 *** Handle the received result data carefully and
                // securely, even though the data came from the Receiver within
                // the same application.
                // Omitted, since this is a sample. Please refer to
                // "3.2 Handling Input Data Carefully and Securely."
                String data = getResultData();
                PrivateSenderActivity.this
                    .logLine(String.format("Received result: \"%s\"", data));
            }
        };

    private TextView mLogView;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
```

(continues on next page)

(continued from previous page)

```
        setContentView(R.layout.main);
        mLogView = (TextView)findViewById(R.id.logview);
    }

    private void logLine(String line) {
        mLogView.append(line);
        mLogView.append("\n");
    }
}
```

4.2.1.2 Public Broadcast Receiver - Receiving/Sending Broadcasts

Public Broadcast Receiver is the Broadcast Receiver that can receive Broadcasts from unspecified large number of applications, so it's necessary to pay attention that it may receive Broadcasts from malware.

Points (Receiving Broadcasts):

1. Explicitly set the exported attribute to true.
2. Handle the received Intent carefully and securely.
3. When returning a result, do not include sensitive information.

Public Receiver which is the sample code for public Broadcast Receiver can be used both in static Broadcast Receiver and Dynamic Broadcast Receiver.

PublicReceiver.java

```
/*
 * Copyright (C) 2012-2026 Japan Smartphone Security Association
 *
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 *
 *      http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */

package org.jssec.android.broadcast.publicreceiver;

import android.app.Activity;
import android.content.BroadcastReceiver;
import android.content.Context;
import android.content.Intent;
import android.widget.Toast;

public class PublicReceiver extends BroadcastReceiver {

    private static final String MY_BROADCAST_PUBLIC =
        "org.jssec.android.broadcast.MY_BROADCAST_PUBLIC";

    public boolean isDynamic = false;
    private String getName() {
```

(continues on next page)

(continued from previous page)

```
        return isDynamic ? "Public Dynamic Broadcast Receiver" :
                           "Public Static Broadcast Receiver";
    }

    @Override
    public void onReceive(Context context, Intent intent) {

        // *** POINT 2 *** Handle the received Intent carefully and securely.
        // Since this is a public broadcast receiver, the requesting application
        // may be malware.
        // Omitted, since this is a sample. Please refer to
        // "3.2 Handling Input Data Carefully and Securely."
        if (MY_BROADCAST_PUBLIC.equals(intent.getAction())) {
            String param = intent.getStringExtra("PARAM");
            Toast.makeText(context,
                           String.format("%s:\nReceived param: \"%s\"",
                                         getName(), param),
                           Toast.LENGTH_SHORT).show();
        }

        // *** POINT 3 *** When returning a result, do not include sensitive
        // information.
        // Since this is a public broadcast receiver, the requesting application
        // may be malware.
        // If no problem when the information is taken by malware, it can be
        // returned as result.
        setResultCode(Activity.RESULT_OK);
        setResultData(String.format("Not Sensitive Info from %s", getName()));
        abortBroadcast();
    }
}
```

Static Broadcast Receive is defined in AndroidManifest.xml. Note with caution that—depending on the terminal version—reception of implicit Broadcast Intents may be restricted, as in Table 4.2.2.

```
AndroidManifest.xml
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
          >

  <application
      android:icon="@drawable/ic_launcher"
      android:label="@string/app_name"
      android:allowBackup="false" >

    <!-- Public Static Broadcast Receiver -->
    <!-- *** POINT 1 *** Explicitly set the exported attribute to true. -->
    <receiver
        android:name=".PublicReceiver"
        android:exported="true" >
      <intent-filter>
        <action android:name="org.jssec.android.broadcast.MY_BROADCAST_PUBLIC" />
      </intent-filter>
    </receiver>

    <service
```

(continues on next page)

(continued from previous page)

```
        android:name=".DynamicReceiverService"
        android:exported="false" />

    <activity
        android:name=".PublicReceiverActivity"
        android:label="@string/app_name"
        android:exported="true" >
      <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER" />
      </intent-filter>
    </activity>
  </application>
</manifest>
```

In Dynamic Broadcast Receiver, registration/unregistration is executed by calling registerReceiver() or unregisterReceiver() in the program. In order to execute registration/unregistration by button operations, the button is allocated on PublicReceiverActivity. Since the scope of Dynamic Broadcast Receiver Instance is longer than PublicReceiverActivity, it cannot be kept as the member variable of PublicReceiverActivity. In this case, keep the Dynamic Broadcast Receiver Instance as the member variable of DynamicReceiverService, and then start/end DynamicReceiverService from PublicReceiverActivity to register/unregister Dynamic Broadcast Receiver indirectly.

DynamicReceiverService.java

```
/*
 * Copyright (C) 2012-2026 Japan Smartphone Security Association
 *
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 *
 *      http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */

package org.jssec.android.broadcast.publicreceiver;

import android.app.Service;
import android.content.Intent;
import android.content.IntentFilter;
import android.os.IBinder;
import android.widget.Toast;

public class DynamicReceiverService extends Service {

    private static final String MY_BROADCAST_PUBLIC =
        "org.jssec.android.broadcast.MY_BROADCAST_PUBLIC";

    private PublicReceiver mReceiver;

    @Override
    public IBinder onBind(Intent intent) {
```

(continues on next page)

(continued from previous page)

```
        return null;
    }

    @Override
    public void onCreate() {
        super.onCreate();

        // Register Public Dynamic Broadcast Receiver.
        mReceiver = new PublicReceiver();
        mReceiver.isDynamic = true;
        IntentFilter filter = new IntentFilter();
        filter.addAction(MY_BROADCAST_PUBLIC);
        // Prioritize Dynamic Broadcast Receiver,
        // rather than Static Broadcast Receiver.
        filter.setPriority(1);
        registerReceiver(mReceiver, filter);
        Toast.makeText(this,
                       "Registered Dynamic Broadcast Receiver.",
                       Toast.LENGTH_SHORT).show();
    }

    @Override
    public void onDestroy() {
        super.onDestroy();

        // Unregister Public Dynamic Broadcast Receiver.
        unregisterReceiver(mReceiver);
        mReceiver = null;
        Toast.makeText(this,
                       "Unregistered Dynamic Broadcast Receiver.",
                       Toast.LENGTH_SHORT).show();
    }
}
```

PublicReceiverActivity.java

```
/*
 * Copyright (C) 2012-2026 Japan Smartphone Security Association
 *
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 *
 *      http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */

package org.jssec.android.broadcast.publicreceiver;

import android.app.Activity;
import android.content.Intent;
import android.os.Bundle;
```

(continues on next page)

(continued from previous page)

```
import android.view.View;

public class PublicReceiverActivity extends Activity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
    }

    public void onRegisterReceiverClick(View view) {
        Intent intent = new Intent(this, DynamicReceiverService.class);
        startService(intent);
    }

    public void onUnregisterReceiverClick(View view) {
        Intent intent = new Intent(this, DynamicReceiverService.class);
        stopService(intent);
    }
}
```

Next, the sample code for sending Broadcasts to public Broadcast Receiver is shown. When sending Broadcasts to public Broadcast Receiver, it's necessary to pay attention that Broadcasts can be received by malware.

Points (Sending Broadcasts):

4. Do not send sensitive information.
5. When receiving a result, handle the result data carefully and securely.

PublicSenderActivity.java

```
/*
 * Copyright (C) 2012-2026 Japan Smartphone Security Association
 *
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 *
 *      http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */

package org.jssec.android.broadcast.publicsender;

import android.app.Activity;
import android.content.BroadcastReceiver;
import android.content.Context;
import android.content.Intent;
import android.os.Bundle;
import android.view.View;
import android.widget.TextView;

public class PublicSenderActivity extends Activity {
```

(continues on next page)

(continued from previous page)

```
    private static final String MY_BROADCAST_PUBLIC =
        "org.jssec.android.broadcast.MY_BROADCAST_PUBLIC";

    public void onSendNormalClick(View view) {
        // *** POINT 4 *** Do not send sensitive information.
        Intent intent = new Intent(MY_BROADCAST_PUBLIC);
        intent.putExtra("PARAM", "Not Sensitive Info from Sender");
        sendBroadcast(intent);
    }

    public void onSendOrderedClick(View view) {
        // *** POINT 4 *** Do not send sensitive information.
        Intent intent = new Intent(MY_BROADCAST_PUBLIC);
        intent.putExtra("PARAM", "Not Sensitive Info from Sender");
        sendOrderedBroadcast(intent, null, mResultReceiver, null, 0, null, null);
    }

    public void onSendStickyClick(View view) {
        // *** POINT 4 *** Do not send sensitive information.
        Intent intent = new Intent(MY_BROADCAST_PUBLIC);
        intent.putExtra("PARAM", "Not Sensitive Info from Sender");
        //sendStickyBroadcast is deprecated at API Level 21
        sendStickyBroadcast(intent);
    }

    public void onSendStickyOrderedClick(View view) {
        // *** POINT 4 *** Do not send sensitive information.
        Intent intent = new Intent(MY_BROADCAST_PUBLIC);
        intent.putExtra("PARAM", "Not Sensitive Info from Sender");
        //sendStickyOrderedBroadcast is deprecated at API Level 21
        sendStickyOrderedBroadcast(intent, mResultReceiver, null, 0, null, null);
    }

    public void onRemoveStickyClick(View view) {
        Intent intent = new Intent(MY_BROADCAST_PUBLIC);
        //removeStickyBroadcast is deprecated at API Level 21
        removeStickyBroadcast(intent);
    }

    private BroadcastReceiver mResultReceiver = new BroadcastReceiver() {
            @Override
            public void onReceive(Context context, Intent intent) {

                // *** POINT 5 *** When receiving a result, handle the result data
                // carefully and securely.
                // Omitted, since this is a sample. Please refer to
                // "3.2 Handling Input Data Carefully and Securely."
                String data = getResultData();
                PublicSenderActivity.this
                    .logLine(String.format("Received result: \"%s\"", data));
            }
        };

    private TextView mLogView;
```

(continues on next page)

(continued from previous page)

```
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        mLogView = (TextView)findViewById(R.id.logview);
    }

    private void logLine(String line) {
        mLogView.append(line);
        mLogView.append("\n");
    }
}
```

4.2.1.3 In-house Broadcast Receiver - Receiving/Sending Broadcasts

In-house Broadcast Receiver is the Broadcast Receiver that will never receive any Broadcasts sent from other than in-house applications. It consists of several in-house applications, and it's used to protect the information or functions that in-house application handles.

Points (Receiving Broadcasts):

1. Define an in-house signature permission to receive Broadcasts.
2. Declare to use the in-house signature permission to receive results.
3. Explicitly set the exported attribute to true.
4. Require the in-house signature permission by the Static Broadcast Receiver definition.
5. Require the in-house signature permission to register Dynamic Broadcast Receiver.
6. Verify that the in-house signature permission is defined by an in-house application.
7. Handle the received intent carefully and securely, even though the Broadcast was sent from an in-house application.
8. Sensitive information can be returned since the requesting application is in-house.
9. When Exporting an APK, sign the APK with the same developer key as the sending application.

In-house Receiver which is a sample code of in-house Broadcast Receiver is to be used both in Static Broadcast Receiver and Dynamic Broadcast Receiver.

```
InhouseReceiver.java
/*
 * Copyright (C) 2012-2026 Japan Smartphone Security Association
 *
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 *
 *      http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */

package org.jssec.android.broadcast.inhousereceiver;
```

(continues on next page)

(continued from previous page)

```
import org.jssec.android.shared.SigPerm;
import org.jssec.android.shared.Utils;

import android.app.Activity;
import android.content.BroadcastReceiver;
import android.content.Context;
import android.content.Intent;
import android.widget.Toast;

public class InhouseReceiver extends BroadcastReceiver {

    // In-house Signature Permission
    private static final String MY_PERMISSION =
            "org.jssec.android.broadcast.inhousereceiver.MY_PERMISSION";

    // In-house certificate hash value
    private static String sMyCertHash = null;
    private static String myCertHash(Context context) {
        if (sMyCertHash == null) {
            if (Utils.isDebuggable(context)) {
                // Certificate hash value of "androiddebugkey" in the
                // debug.keystore.
                sMyCertHash = "0EFB7236 328348A9 89718BAD DF57F544 D5CCB4AE␣
↪B9DB34BC 1E29DD26 F77C8255";
            } else {
                // Certificate hash value of "my company key" in the keystore.
                sMyCertHash = "D397D343 A5CBC10F 4EDDEB7C A10062DE 5690984F␣
↪1FB9E88B D7B3A7C2 42E142CA";
            }
        }
        return sMyCertHash;
    }

    private static final String MY_BROADCAST_INHOUSE =
        "org.jssec.android.broadcast.MY_BROADCAST_INHOUSE";

    public boolean isDynamic = false;
    private String getName() {
        return isDynamic ? "In-house Dynamic Broadcast Receiver" :
                           "In-house Static Broadcast Receiver";
    }

    @Override
    public void onReceive(Context context, Intent intent) {

        // *** POINT 6 *** Verify that the in-house signature permission is
        // defined by an in-house application.
        if (!SigPerm.test(context, MY_PERMISSION, myCertHash(context))) {
            Toast.makeText(context,"The in-house signature permission is not␣
↪declared by in-house application.", Toast.LENGTH_LONG).show();
            return;
        }

        // *** POINT 7 *** Handle the received intent carefully and securely,
        // even though the Broadcast was sent from an in-house application..
```

(continues on next page)

(continued from previous page)

```
        // Omitted, since this is a sample. Please refer to
        // "3.2 Handling Input Data Carefully and Securely."
        if (MY_BROADCAST_INHOUSE.equals(intent.getAction())) {
            String param = intent.getStringExtra("PARAM");
            Toast.makeText(context, String.format("%s:\nReceived param: \"%s\"",␣
↪getName(), param), Toast.LENGTH_SHORT).show();
        }

        // *** POINT 8 *** Sensitive information can be returned since the
        // requesting application is in-house.
        setResultCode(Activity.RESULT_OK);
        setResultData(String.format("Sensitive Info from %s", getName()));
        abortBroadcast();
    }
}
```

Static Broadcast Receiver is to be defined in AndroidManifest.xml.Note with caution that—depending on the terminal version—reception of implicit Broadcast Intents may be restricted, as in Table 4.2.2.

AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
          >

  <!-- *** POINT 1 *** Define an in-house signature permission to receive␣
↪Broadcasts -->
  <permission
      android:name="org.jssec.android.broadcast.inhousereceiver.MY_PERMISSION"
      android:protectionLevel="signature" />

  <!-- *** POINT 2 *** Declare to use the in-house signature permission to receive␣
↪results. -->
  <uses-permission
      android:name="org.jssec.android.broadcast.inhousesender.MY_PERMISSION" />

  <application
      android:icon="@drawable/ic_launcher"
      android:label="@string/app_name"
      android:allowBackup="false" >

    <!-- *** POINT 3 *** Explicitly set the exported attribute to true. -->
    <!-- *** POINT 4 *** Require the in-house signature permission by the Static␣
↪Broadcast Receiver definition. -->
    <receiver
        android:name=".InhouseReceiver"
        android:permission="org.jssec.android.broadcast.inhousereceiver.MY_
↪PERMISSION"
        android:exported="true">
      <intent-filter>
        <action android:name="org.jssec.android.broadcast.MY_BROADCAST_INHOUSE" />
      </intent-filter>
    </receiver>

    <service
        android:name=".DynamicReceiverService"
        android:exported="false" />
```

(continues on next page)

(continued from previous page)

```
    <activity
        android:name=".InhouseReceiverActivity"
        android:label="@string/app_name"
        android:exported="true" >
      <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER" />
      </intent-filter>
    </activity>
  </application>

</manifest>
```

Dynamic Broadcast Receiver executes registration/unregistration by calling registerReceiver() or unregisterReceiver() in the program. In order to execute registration/unregistration by the button operations, the button is arranged on InhouseReceiverActivity. Since the scope of Dynamic Broadcast Receiver Instance is longer than InhouseReceiverActivity, it cannot be kept as the member variable of InhouseReceiverActivity. So, keep Dynamic Broadcast Receiver Instance as the member variable of DynamicReceiverService, and then start/end DynamicReceiverService from InhouseReceiverActivity to register/unregister Dynamic Broadcast Receiver indirectly.

InhouseReceiverActivity.java

```
/*
 * Copyright (C) 2012-2026 Japan Smartphone Security Association
 *
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 *
 *      http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */

package org.jssec.android.broadcast.inhousereceiver;

import android.app.Activity;
import android.content.Intent;
import android.os.Bundle;
import android.view.View;

public class InhouseReceiverActivity extends Activity {
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
    }

    public void onRegisterReceiverClick(View view) {
        Intent intent = new Intent(this, DynamicReceiverService.class);
        startService(intent);
    }
```

(continues on next page)

(continued from previous page)

```
    public void onUnregisterReceiverClick(View view) {
        Intent intent = new Intent(this, DynamicReceiverService.class);
        stopService(intent);
    }
}
```

DynamicReceiverService.java

```
/*
 * Copyright (C) 2012-2026 Japan Smartphone Security Association
 *
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 *
 *      http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */

package org.jssec.android.broadcast.inhousereceiver;

import android.app.Service;
import android.content.Intent;
import android.content.IntentFilter;
import android.os.IBinder;
import android.widget.Toast;

public class DynamicReceiverService extends Service {

    private static final String MY_BROADCAST_INHOUSE =
        "org.jssec.android.broadcast.MY_BROADCAST_INHOUSE";

    private InhouseReceiver mReceiver;

    @Override
    public IBinder onBind(Intent intent) {
        return null;
    }

    @Override
    public void onCreate() {
        super.onCreate();

        mReceiver = new InhouseReceiver();
        mReceiver.isDynamic = true;
        IntentFilter filter = new IntentFilter();
        filter.addAction(MY_BROADCAST_INHOUSE);
        // Prioritize Dynamic Broadcast Receiver,
        // rather than Static Broadcast Receiver.
        filter.setPriority(1);
```

(continues on next page)

(continued from previous page)

```
        // *** POINT 5 *** When registering a dynamic broadcast receiver, require
        // the in-house signature permission.
        registerReceiver(mReceiver, filter,
                         "org.jssec.android.broadcast.inhousereceiver.MY_PERMISSION
↪",
                         null);

        Toast.makeText(this,
                       "Registered Dynamic Broadcast Receiver.",
                       Toast.LENGTH_SHORT).show();
    }

    @Override
    public void onDestroy() {
        super.onDestroy();
        unregisterReceiver(mReceiver);
        mReceiver = null;
        Toast.makeText(this,
                       "Unregistered Dynamic Broadcast Receiver.",
                       Toast.LENGTH_SHORT).show();
    }
}
```

SigPerm.java

```
/*
 * Copyright (C) 2012-2026 Japan Smartphone Security Association
 *
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 *
 *      http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */

package org.jssec.android.shared;

import android.content.Context;
import android.content.pm.PackageManager;
import android.content.pm.PackageManager.NameNotFoundException;
import android.content.pm.PermissionInfo;
import android.os.Build;

import static android.content.pm.PackageManager.CERT_INPUT_SHA256;

public class SigPerm {

    public static boolean test(Context ctx, String sigPermName,
                               String correctHash) {
        if (correctHash == null) return false;
        correctHash = correctHash.replaceAll(" ", "");
```

(continues on next page)

(continued from previous page)

```
        try {
            // Get the package name of the application which declares a permission
            // named sigPermName.
            PackageManager pm = ctx.getPackageManager();
            PermissionInfo pi =
                pm.getPermissionInfo(sigPermName, PackageManager.GET_META_DATA);
            String pkgname = pi.packageName;
            // Fail if the permission named sigPermName is not a Signature
            // Permission
            if (pi.protectionLevel != PermissionInfo.PROTECTION_SIGNATURE)
                return false;

            // Compare the actual hash value of pkgname with the correct hash
            // value.
            if (Build.VERSION.SDK_INT >= 28) {
                // ** if API Level >= 28, direct check is possible
                return pm.hasSigningCertificate(pkgname,
                                                Utils.hex2Bytes(correctHash),
                                                CERT_INPUT_SHA256);
            } else {
                // else(API Level < 28) use the facility of PkgCert
                return correctHash.equals(PkgCert.hash(ctx, pkgname));
            }

        } catch (NameNotFoundException e) {
            return false;
        }
    }
}
```

PkgCert.java

```
/*
 * Copyright (C) 2012-2026 Japan Smartphone Security Association
 *
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 *
 *      http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */

package org.jssec.android.shared;

import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;

import android.content.Context;
import android.content.pm.PackageInfo;
import android.content.pm.PackageManager;
import android.content.pm.PackageManager.NameNotFoundException;
```

(continues on next page)

(continued from previous page)

```
import android.content.pm.Signature;

public class PkgCert {

    public static boolean test(Context ctx, String pkgname, String correctHash) {
        if (correctHash == null) return false;
        correctHash = correctHash.replaceAll(" ", "");
        return correctHash.equals(hash(ctx, pkgname));
    }

    public static String hash(Context ctx, String pkgname) {
        if (pkgname == null) return null;
        try {
            PackageManager pm = ctx.getPackageManager();
            PackageInfo pkginfo =
                pm.getPackageInfo(pkgname, PackageManager.GET_SIGNATURES);
            // Will not handle multiple signatures.
            if (pkginfo.signatures.length != 1) return null;
            Signature sig = pkginfo.signatures[0];
            byte[] cert = sig.toByteArray();
            byte[] sha256 = computeSha256(cert);
            return byte2hex(sha256);
        } catch (NameNotFoundException e) {
            return null;
        }
    }

    private static byte[] computeSha256(byte[] data) {
        try {
            return MessageDigest.getInstance("SHA-256").digest(data);
        } catch (NoSuchAlgorithmException e) {
            return null;
        }
    }

    private static String byte2hex(byte[] data) {
        if (data == null) return null;
        final StringBuilder hexadecimal = new StringBuilder();
        for (final byte b : data) {
            hexadecimal.append(String.format("%02X", b));
        }
        return hexadecimal.toString();
    }
}
```

*** Point 9 *** When exporting an APK, sign the APK with the same developer key as the sending application.

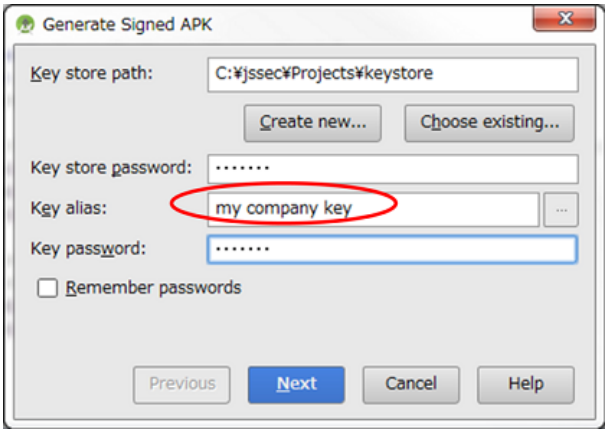


Fig. 4.2.2: Sign the APK with the same developer key as the sending application

Next, the sample code for sending Broadcasts to in-house Broadcast Receiver is shown.

Points (Sending Broadcasts):

10. Define an in-house signature permission to receive results.
11. Declare to use the in-house signature permission to receive Broadcasts.
12. Verify that the in-house signature permission is defined by an in-house application.
13. Sensitive information can be returned since the requesting application is the in-house one.
14. Require the in-house signature permission of Receivers.
15. Handle the received result data carefully and securely.
16. When exporting an APK, sign the APK with the same developer key as the destination application.

```
AndroidManifest.xml
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
          >

  <queries>
    <package android:name="org.jssec.android.broadcast.inhousereceiver" />
  </queries>

  <uses-permission android:name="android.permission.BROADCAST_STICKY"/>

  <!-- *** POINT 10 *** Define an in-house signature permission to receive results.
↪ -->
  <permission
      android:name="org.jssec.android.broadcast.inhousesender.MY_PERMISSION"
      android:protectionLevel="signature" />

  <!-- *** POINT 11 *** Declare to use the in-house signature permission to␣
↪receive Broadcasts. -->
  <uses-permission
      android:name="org.jssec.android.broadcast.inhousereceiver.MY_PERMISSION" />

  <application
      android:icon="@drawable/ic_launcher"
      android:label="@string/app_name"
      android:allowBackup="false" >
    <activity
```

(continues on next page)

(continued from previous page)

```
        android:name="org.jssec.android.broadcast.inhousesender.
↪InhouseSenderActivity"
        android:label="@string/app_name"
        android:exported="true" >
      <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER" />
      </intent-filter>
    </activity>
  </application>
</manifest>
```

InhouseSenderActivity.java

```
/*
 * Copyright (C) 2012-2026 Japan Smartphone Security Association
 *
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 *
 *      http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */

package org.jssec.android.broadcast.inhousesender;

import org.jssec.android.shared.SigPerm;
import org.jssec.android.shared.Utils;

import android.app.Activity;
import android.content.BroadcastReceiver;
import android.content.Context;
import android.content.Intent;
import android.os.Bundle;
import android.view.View;
import android.widget.TextView;
import android.widget.Toast;

public class InhouseSenderActivity extends Activity {

    // In-house Signature Permission
    private static final String MY_PERMISSION =
            "org.jssec.android.broadcast.inhousesender.MY_PERMISSION";

    // In-house certificate hash value
    private static String sMyCertHash = null;
    private static String myCertHash(Context context) {
        if (sMyCertHash == null) {
            if (Utils.isDebuggable(context)) {
                // Certificate hash value of "androiddebugkey" in the
                // debug.keystore.
```

(continues on next page)

(continued from previous page)

```
                sMyCertHash = "0EFB7236 328348A9 89718BAD DF57F544 D5CCB4AE␣
↪B9DB34BC 1E29DD26 F77C8255";
            } else {
                // Certificate hash value of "my company key" in the keystore.
                sMyCertHash = "D397D343 A5CBC10F 4EDDEB7C A10062DE 5690984F␣
↪1FB9E88B D7B3A7C2 42E142CA";
            }
        }
        return sMyCertHash;
    }

    private static final String MY_BROADCAST_INHOUSE =
        "org.jssec.android.broadcast.MY_BROADCAST_INHOUSE";

    public void onSendNormalClick(View view) {

        // *** POINT 12 *** Verify that the in-house signature permission is
        // defined by an in-house application.
        if (!SigPerm.test(this, MY_PERMISSION, myCertHash(this))) {
            Toast.makeText(this, "The in-house signature permission is not␣
↪declared by in-house application.", Toast.LENGTH_LONG).show();
            return;
        }

        // *** POINT 13 *** Sensitive information can be returned since the
        // requesting application is in-house.
        Intent intent = new Intent(MY_BROADCAST_INHOUSE);
        intent.putExtra("PARAM", "Sensitive Info from Sender");

        // *** POINT 14 *** Require the in-house signature permission to limit
        // receivers.
        sendBroadcast(intent,
                      "org.jssec.android.broadcast.inhousesender.MY_PERMISSION");
    }

    public void onSendOrderedClick(View view) {

        // *** POINT 12 *** Verify that the in-house signature permission is
        // defined by an in-house application.
        if (!SigPerm.test(this, MY_PERMISSION, myCertHash(this))) {
            Toast.makeText(this, "The in-house signature permission is not␣
↪declared by in-house application.", Toast.LENGTH_LONG).show();
            return;
        }

        // *** POINT 13 *** Sensitive information can be returned since the
        // requesting application is in-house.
        Intent intent = new Intent(MY_BROADCAST_INHOUSE);
        intent.putExtra("PARAM", "Sensitive Info from Sender");

        // *** POINT 14 *** Require the in-house signature permission to limit
        // receivers.
        sendOrderedBroadcast(intent,
                "org.jssec.android.broadcast.inhousesender.MY_PERMISSION",
                mResultReceiver, null, 0, null, null);
    }
```

(continues on next page)

(continued from previous page)

```
    private BroadcastReceiver mResultReceiver = new BroadcastReceiver() {
            @Override
            public void onReceive(Context context, Intent intent) {

                // *** POINT 15 *** Handle the received result data carefully and
                // securely, even though the data came from an in-house
                // application.
                // Omitted, since this is a sample. Please refer to
                // "3.2 Handling Input Data Carefully and Securely."
                String data = getResultData();
                InhouseSenderActivity.this
                    .logLine(String.format("Received result: \"%s\"", data));
            }
        };

    private TextView mLogView;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        mLogView = (TextView)findViewById(R.id.logview);
    }

    private void logLine(String line) {
        mLogView.append(line);
        mLogView.append("\n");
    }
}
```

SigPerm.java

```
/*
 * Copyright (C) 2012-2026 Japan Smartphone Security Association
 *
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 *
 *      http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */

package org.jssec.android.shared;

import android.content.Context;
import android.content.pm.PackageManager;
import android.content.pm.PackageManager.NameNotFoundException;
import android.content.pm.PermissionInfo;
import android.os.Build;
```

(continues on next page)

(continued from previous page)

```
import static android.content.pm.PackageManager.CERT_INPUT_SHA256;

public class SigPerm {

    public static boolean test(Context ctx, String sigPermName,
                               String correctHash) {
        if (correctHash == null) return false;
        correctHash = correctHash.replaceAll(" ", "");
        try {
            // Get the package name of the application which declares a permission
            // named sigPermName.
            PackageManager pm = ctx.getPackageManager();
            PermissionInfo pi =
                pm.getPermissionInfo(sigPermName, PackageManager.GET_META_DATA);
            String pkgname = pi.packageName;
            // Fail if the permission named sigPermName is not a Signature
            // Permission
            if (pi.protectionLevel != PermissionInfo.PROTECTION_SIGNATURE)
                return false;

            // Compare the actual hash value of pkgname with the correct hash
            // value.
            if (Build.VERSION.SDK_INT >= 28) {
                // ** if API Level >= 28, direct check is possible
                return pm.hasSigningCertificate(pkgname,
                                                Utils.hex2Bytes(correctHash),
                                                CERT_INPUT_SHA256);
            } else {
                // else(API Level < 28) use the facility of PkgCert
                return correctHash.equals(PkgCert.hash(ctx, pkgname));
            }

        } catch (NameNotFoundException e) {
            return false;
        }
    }
}
```

PkgCert.java

```
/*
 * Copyright (C) 2012-2026 Japan Smartphone Security Association
 *
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 *
 *      http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */

package org.jssec.android.shared;
```

(continues on next page)

(continued from previous page)

```
import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;

import android.content.Context;
import android.content.pm.PackageInfo;
import android.content.pm.PackageManager;
import android.content.pm.PackageManager.NameNotFoundException;
import android.content.pm.Signature;

public class PkgCert {

    public static boolean test(Context ctx, String pkgname, String correctHash) {
        if (correctHash == null) return false;
        correctHash = correctHash.replaceAll(" ", "");
        return correctHash.equals(hash(ctx, pkgname));
    }

    public static String hash(Context ctx, String pkgname) {
        if (pkgname == null) return null;
        try {
            PackageManager pm = ctx.getPackageManager();
            PackageInfo pkginfo =
                pm.getPackageInfo(pkgname, PackageManager.GET_SIGNATURES);
            // Will not handle multiple signatures.
            if (pkginfo.signatures.length != 1) return null;
            Signature sig = pkginfo.signatures[0];
            byte[] cert = sig.toByteArray();
            byte[] sha256 = computeSha256(cert);
            return byte2hex(sha256);
        } catch (NameNotFoundException e) {
            return null;
        }
    }

    private static byte[] computeSha256(byte[] data) {
        try {
            return MessageDigest.getInstance("SHA-256").digest(data);
        } catch (NoSuchAlgorithmException e) {
            return null;
        }
    }

    private static String byte2hex(byte[] data) {
        if (data == null) return null;
        final StringBuilder hexadecimal = new StringBuilder();
        for (final byte b : data) {
            hexadecimal.append(String.format("%02X", b));
        }
        return hexadecimal.toString();
    }
}
```

*** Point 16 *** When exporting an APK, sign the APK with the same developer key as the destination application.

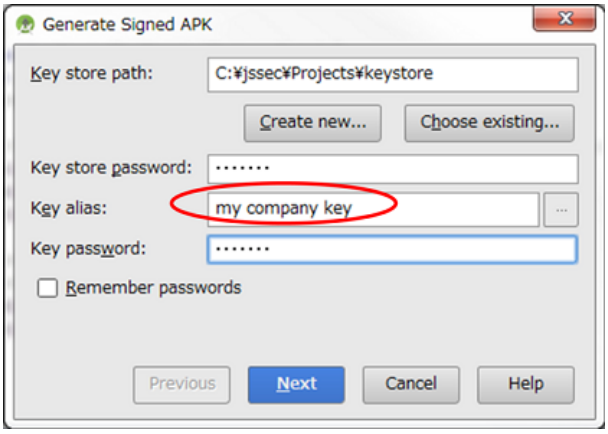


Fig. 4.2.3: Sign the APK with the same developer key as the destination application

4.2.2 Rule Book

Follow the rules below to Send or receive Broadcasts.

1. *Broadcast Receiver that Is Used Only in an Application Must Be Set as Private (Required)*
2. *Handle the Received Intent Carefully and Securely (Required)*
3. *Use the In-house Defined Signature Permission after Verifying that it's Defined by an In-house Application (Required)*
4. *When Returning a Result Information, Pay Attention to the Result Information Leakage from the Destination Application (Required)*
5. *When Sending Sensitive Information with a Broadcast, Limit the Receivable Receiver (Required)*
6. *Sensitive Information Must Not Be Included in the Sticky Broadcast (Required)*
7. *Pay Attention that the Ordered Broadcast without Specifying the receiverPermission May Not Be Delivered (Required)*
8. *Handle the Returned Result Data from the Broadcast Receiver Carefully and Securely (Required)*
9. *When Providing an Asset Secondarily, the Asset should be protected with the Same Protection Level (Required)*

4.2.2.1 Broadcast Receiver that Is Used Only in an Application Must Be Set as Private (Required)

Broadcast Receiver which is used only in the application should be set as private to avoid from receiving any Broadcasts from other applications unexpectedly. It will prevent the application function abuse or the abnormal behaviors.

Receiver used only within the same application should not be designed with setting Intent-filter. Because of the Intent-filter characteristics, a public Receiver of other application may be called unexpectedly by calling through Intent-filter even though a private Receiver within the same application is to be called.

```
AndroidManifest.xml(Not recoomended)
      <!-- Private Broadcast Receiver -->
      <!-- *** 4.2.1.1 - POINT 1 *** Explicitly set the exported attribute to␣
↪false. -->
      <receiver android:name=".PrivateReceiver"
          android:exported="false" >
          <intent-filter>
              <action android:name="org.jssec.android.broadcast.MY_ACTION" />
          </intent-filter>
       </receiver>
```

Please refer to "4.2.3.1. *Combination of Exported Attribute and the Intent-filter setting (For Receiver)*."

4.2.2.2 Handle the Received Intent Carefully and Securely (Required)

Though risks are different depending on the types of the Broadcast Receiver, firstly verify the safety of Intent when processing received Intent data.

Since Public Broadcast Receiver receives the Intents from unspecified large number of applications, it may receive malware's attacking Intents. Private Broadcast Receiver will never receive any Intent from other applications directly, but Intent data which a public Component received from other applications may be forwarded to Private Broadcast Receiver. So don't think that the received Intent is totally safe without any qualification. In-house Broadcast Receivers have some degree of the risks, so it also needs to verify the safety of the received Intents.

Please refer to "3.2. *Handling Input Data Carefully and Securely*"

4.2.2.3 Use the In-house Defined Signature Permission after Verifying that it's Defined by an In-house Application (Required)

In-house Broadcast Receiver which receives only Broadcasts sent by an In-house application should be protected by in-house-defined Signature Permission. Permission definition/Permission request declarations in AndroidManifest.xml are not enough to protecting, so please refer to "5.2.1.2. *How to Communicate Between In-house Applications with In-house-defined Signature Permission*." ending Broadcasts by specifying in-house-defined Signature Permission to receiverPermission parameter requires verification in the same way.

4.2.2.4 When Returning a Result Information, Pay Attention to the Result Information Leakage from the Destination Application (Required)

The Reliability of the application which returns result information by setResult() varies depending on the types of the Broadcast Receiver. In case of Public Broadcast Receiver, the destination application may be malware, and there may be a risk that the result information is used maliciously. In case of Private Broadcast Receiver and In-house Broadcast Receiver, the result destination is In-house developed application, so no need to mind the result information handling.

Need to pay attention to the result information leakage from the destination application when result information is returned from Broadcast Receivers as above.

4.2.2.5 When Sending Sensitive Information with a Broadcast, Limit the Receivable Receiver (Required)

Broadcast is the created system to broadcast information to unspecified large number of applications or notify them of the timing at once. So, broadcasting sensitive information requires the careful designing for preventing the illicit obtainment of the information by malware.

For broadcasting sensitive information, only reliable Broadcast Receiver can receive it, and other Broadcast Receivers cannot. The following are some examples of Broadcast sending methods.

- The method is to fix the address by Broadcast-sending with an explicit Intent for sending Broadcasts to the intended reliable Broadcast Receivers only. There are 2 patterns in this method.
  - When it's addressed to a Broadcast Receiver within the same application, specify the address by Intent#setClass(Context, Class). Refer to sample code section "4.2.1.1. *Private Broadcast Receiver - Receiving/Sending Broadcasts*" for the concrete code.
  - When it's addressed to a Broadcast Receiver in other applications, specify the address by Intent#setClassName(String, String). Confirm the permitted application by comparing the developer key of the APK signature in the destination package with the white list to send Broadcasts. Actually the following method of using implicit Intents is more practical.

- The Method is to send Broadcasts by specifying in-house-defined Signature Permission to receiverPermission parameter and make the reliable Broadcast Receiver declare to use this Signature Permission. Refer to the sample code section "4.2.1.3. *In-house Broadcast Receiver - Receiving/Sending Broadcasts*" for the concrete code. In addition, implementing this Broadcast-sending method needs to apply the rule "4.2.2.3. *Use the In-house Defined Signature Permission after Verifying that it's Defined by an In-house Application (Required)*."

4.2.2.6 Sensitive Information Must Not Be Included in the Sticky Broadcast (Required)

Usually, the Broadcasts will be disappeared when they are processed to be received by the available Broadcast Receivers. On the other hand, Sticky Broadcasts (hereafter, Sticky Broadcasts including Sticky Ordered Broadcasts), will not be disappeared from the system even when they processed to be received by the available Broadcast Receivers and will be able to be received by registerReceiver(). When Sticky Broadcast becomes unnecessary, it can be deleted anytime arbitrarily with removeStickyBroadcast().

As it's presupposed that Sticky Broadcast is used by the implicit Intent. Broadcasts with specified receiverPermission Parameter cannot be sent. For this reason, information sent via Sticky Broadcasts can be accessed by multiple unspecified apps — including malware — and thus sensitive information must not be sent in this way. Note that Sticky Broadcast is deprecated in Android 5.0 (API Level 21).

4.2.2.7 Pay Attention that the Ordered Broadcast without Specifying the receiverPermission May Not Be Delivered (Required)

Ordered Broadcast without specified receiverPermission Parameter can be received by unspecified large number of applications including malware. Ordered Broadcast is used to receive the returned information from Receiver, and to make several Receivers execute processing one by one. Broadcasts are sent to the Receivers in order of priority. So if the high- priority malware receives Broadcast first and executes abortBroadcast(), Broadcasts won't be delivered to the following Receivers.

4.2.2.8 Handle the Returned Result Data from the Broadcast Receiver Carefully and Securely (Required)

Basically the result data should be processed safely considering the possibility that received results may be the attacking data though the risks vary depending on the types of the Broadcast Receiver which has returned the result data.

When sender (source) Broadcast Receiver is public Broadcast Receiver, it receives the returned data from unspecified large number of applications. So it may also receive malware's attacking data. When sender (source) Broadcast Receiver is private Broadcast Receiver, it seems no risk. However the data received by other applications may be forwarded as result data indirectly. So the result data should not be considered as safe without any qualification. When sender (source) Broadcast Receiver is In-house Broadcast Receiver, it has some degree of the risks. So it should be processed in a safe way considering the possibility that the result data may be an attacking data.

Please refer to "3.2. *Handling Input Data Carefully and Securely*"

4.2.2.9 When Providing an Asset Secondarily, the Asset should be protected with the Same Protection Level (Required)

When information or function assets protected by Permission are provided to other applications secondarily, it's necessary to keep the protection standard by claiming the same Permission of the destination application. In the Android Permission security models, privileges are managed only for the direct access to the protected assets from applications. Because of the characteristics, acquired assets may be provided to other applications without claiming Permission which is necessary for protection. This is actually same as re-delegating Permission, as it is called, Permission re-delegation problem. Please refer to "5.2.3.4. *Permission Re-delegation Problem*."

4.2.3 Advanced Topics

4.2.3.1 Combination of Exported Attribute and the Intent-filter setting (For Receiver)

Table 4.2.3 represents the permitted combination of export settings and Intent-filter elements when implementing Receivers. The reason why the usage of "exported="false" with Intent-filter definition" is principally prohibited, is described below.

Table 4.2.3: Combination of exported attribute setting and intent-filter setting

Value of exported attribute	intent-filter defined	intent-filter not defined
true	Allowed	Allowed
false	Prohibited	Allowed
Unspecified	Generally prohibited	Generally prohibited

When the exported attribute of a Receiver is left unspecified, the question of whether or not the Receiver is public is determined by the presence or absence of intent filters for that Receiver[11]. However, in this guidebook it is forbidden to set the exported attribute to "unspecified". In general, as mentioned previously, it is best to avoid implementations that rely on the default behavior of any given API; moreover, in cases where explicit methods — such as the exported attribute — exist for enabling important security-related settings, it is always a good idea to make use of those methods.

Public Receivers in other applications may be called unexpectedly even though Broadcasts are sent to the private Receivers within the same applications. This is the reason why specifying exported="false" with Intent-filter definition is prohibited. The following 2 figures show how the unexpected calls occur.

Fig. 4.2.4 is an example of the normal behaviors which a private Receiver (application A) can be called by implicit Intent only within the same application. Intent-filter (in the figure, action="X") is defined only in application A, so this is the expected behavior.

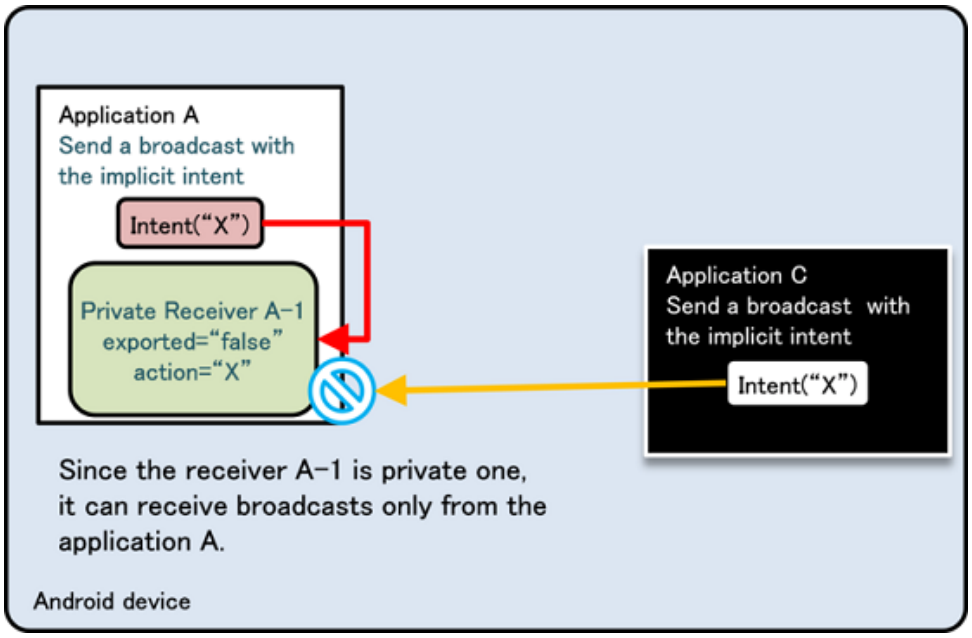


Fig. 4.2.4: An Example of Normal Behavior

Fig. 4.2.5 is an example that Intent-filter (see action="X" in the figure) is defined in the application B as well as in

[11] If any intent filters are defined then the Receiver is public; otherwise it is private. For more information, see https://developer.android.com/guide/topics/manifest/receiver-element.html#exported

the application A. First of all, when another application (application C) sends Broadcasts by implicit Intent, they are not received by a private Receiver (A-1) side. So there won't be any security problem. (See the orange arrow marks in the Figure.)

From security point of view, the problem is application A's call to the private Receiver within the same application. When the application A broadcasts implicit Intent, not only Private Receiver within the same application, but also public Receiver (B-1) with the same Intent-filter definition can also receive the Intent. (Red arrow marks in the Figure). In this case, sensitive information may be sent from the application A to B. When the application B is malware, it will cause the leakage of sensitive information. When the Broadcast is Ordered Broadcast, it may receive the unexpected result information.

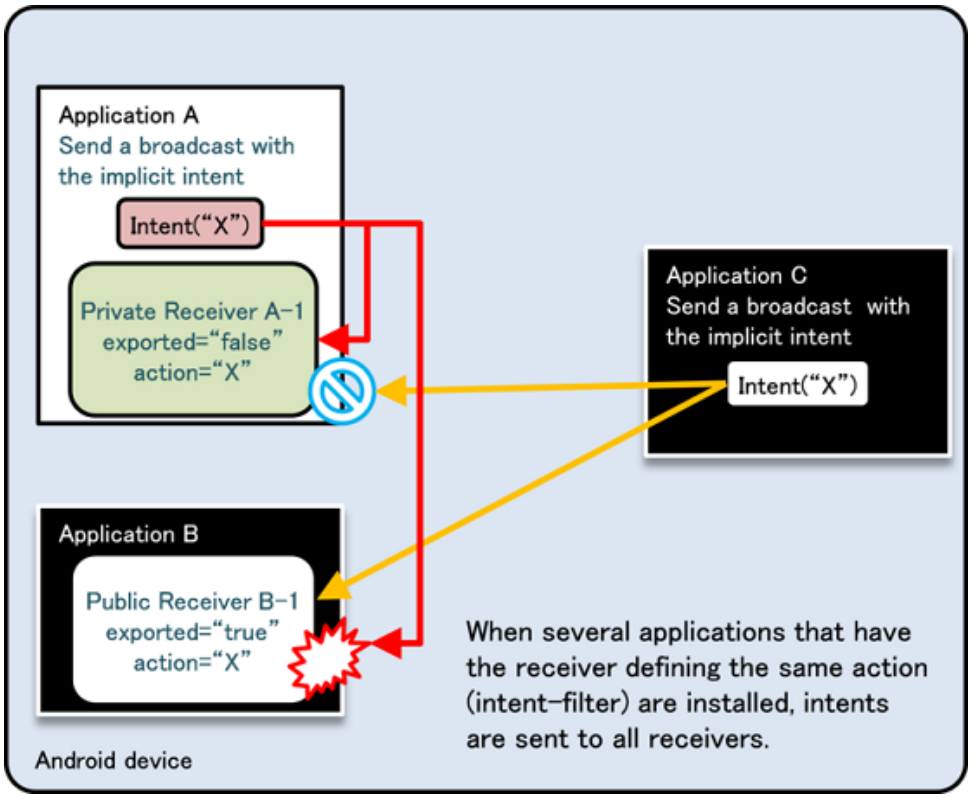


Fig. 4.2.5: An Example of Abnormal Behavior

However, "exported="false" with Intent-filter definition" should be used when Broadcast Receiver to receive only Broadcast Intent sent by the system is implemented. Other combination should not be used. This is based on the fact that Broadcast Intent sent by the system can be received by exported="false". If other applications send Intent which has same ACTION with Broadcast Intent sent by system, it may cause an unexpected behavior by receiving it. However, this can be prevented by specifying exported="false".

Note that from Android 12 onwards, the exported attribute is required. For details, please refer to 4.7.3.1. *Component Export Control and Intent Sending Restrictions*.

4.2.3.2 Receiver Won't Be Registered before Launching the Application

It is important to note carefully that a Broadcast Receiver defined statically in AndroidManifest.xml will not be automatically enabled upon installation[12]. Apps are able to receive Broadcasts only after they have been launched the first time; thus, it is not possible to use the receipt of a Broadcast after installation as a trigger to initiate operations. However, if the Intent.FLAG_INCLUDE_STOPPED_PACKAGES flag set when sending a Broadcast, that Broadcast will be received even by apps that have not yet been launched for the first time.

[12] In versions prior to Android 3.0, Receivers were registered automatically simply by installing apps.

4.2.3.3 Private Broadcast Receiver Can Receive the Broadcast that Was Sent by the Same UID Application

Same UID can be provided to several applications. Even if it's private Broadcast Receiver, the Broadcasts sent from the same UID application can be received.

However, it won't be a security problem. Since it's guaranteed that applications with the same UID have the consistent developer keys for signing APK. It means that what private Broadcast Receiver receives is only the Broadcast sent from In-house applications.

4.2.3.4 Types and Features of Broadcasts

Regarding Broadcasts, there are 4 types based on the combination of whether it's Ordered or not, and Sticky or not. Based on Broadcast sending methods, a type of Broadcast to send is determined. Note that Sticky Broadcast is deprecated in Android 5.0 (API Level 21).

Table 4.2.4: Type of Sending Broadcast

Type of Broadcast	Method for sending	Ordered?	Sticky?
Normal Broadcast	sendBroadcast()	No	No
Ordered Broadcast	sendOrderedBroadcast()	Yes	No
Sticky Broadcast	sendStickyBroadcast()	No	Yes
Sticky Ordered Broadcast	sendStickyOrderedBroadcast()	Yes	Yes

The feature of each Broad cast is described.

Table 4.2.5: Feature of Each Broadcast

Type of Broadcast	Features for each type of Broadcast
Normal Broadcast	Normal Broadcast disappears when it is sent to receivable Broadcast Receiver. Broadcasts are received by several Broadcast Receivers simultaneously. This is a difference from Ordered Broadcast. Broadcasts are allowed to be received by the particular Broadcast Receivers.
Ordered Broadcast	Ordered Broadcast is characterized by receiving Broadcasts one by one in order with receivable Broadcast Receivers. The higher-priority Broadcast Receiver receives earlier. Broadcasts will disappear when Broadcasts are delivered to all Broadcast Receivers or a Broadcast Receiver in the process calls abortBroadcast(). Broadcasts are allowed to be received by the Broadcast Receivers which declare the specified Permission. In addition, the result information sent from Broadcast Receiver can be received by the sender with Ordered Broadcasts. The Broadcast of SMS-receiving notice (SMS_RECEIVED) is a representative example of Ordered Broadcast.
Sticky Broadcast	Sticky Broadcast does not disappear and remains in the system, and then the application that calls registerReceiver() can receive Sticky Broadcast later. Since Sticky Broadcast is different from other Broadcasts, it will never disappear automatically. So when Sticky Broadcast is not necessary, calling removeStickyBroadcast() explicitly is required to delete Sticky Broadcast. Also, Broadcasts cannot be received by the limited Broadcast Receivers with particular Permission. The Broadcast of changing battery-state notice (ACTION_BATTERY_CHANGED) is the representative example of Sticky Broadcast.
Sticky Ordered Broadcast	This is the Broadcast which has both characteristics of Ordered Broadcast and Sticky Broadcast. Same as Sticky Broadcast, it cannot allow only Broadcast Receivers with the particular Permission to receive the Broadcast.

From the Broadcast characteristic behavior point of view, above table is conversely arranged in the following one.

Table 4.2.6: Characteristic behavior of Broadcast

Characteristic behavior of Broadcast	Normal Broadcast	Ordered Broadcast	Sticky Broadcast	Sticky Ordered Broadcast
Limit Broadcast Receivers which can receive Broadcast, by Permission	OK	OK	-	-
Get the results of process from Broadcast Receiver	-	OK	-	OK
Make Broadcast Receivers process Broadcasts in order	-	OK	-	OK
Receive Broadcasts later, which have been already sent	-	-	OK	OK

4.2.3.5 Broadcasted Information May be Output to the LogCat

Basically sending/receiving Broadcasts is not output to LogCat. However, the error log will be output when lacking Permission causes errors in receiver/sender side. Intent information sent by Broadcast is included in the error log, so after an error occurs it's necessary to pay attention that Intent information is displayed in LogCat when Broadcast is sent.

Erorr of lacking Permission in sender side

```
W/ActivityManager(266): Permission Denial: broadcasting Intent {
act=org.jssec.android.broadcastreceiver.creating.action.MY_ACTION }
from org.jssec.android.broadcast.sending (pid=4685, uid=10058) requires
org.jssec.android.permission.MY_PERMISSION due to receiver
org.jssec.android.broadcastreceiver.creating/org.jssec.android.broadcastreceiver.
↪creating.CreatingType3Receiver
```

Erorr of lacking Permission in receiver side

```
W/ActivityManager(266): Permission Denial: broadcasting Intent {
act=org.jssec.android.broadcastreceiver.creating.action.MY_ACTION }
from org.jssec.android.broadcast.sending (pid=4685, uid=10058) requires
org.jssec.android.permission.MY_PERMISSION due to receiver
org.jssec.android.broadcastreceiver.creating/org.jssec.android.broadcastreceiver.
↪creating.CreatingType3Receiver
```

4.2.3.6 Items to Keep in Mind When Placing an App Shortcut on the Home Screen

In what follows we discuss a number of items to keep in mind when creating a shortcut button for launching an app from the home screen or for creating URL shortcuts such as bookmarks in web browsers. As an example, we consider the implementation shown below.

Place an app shortcut on the home screen

```
Intent targetIntent = new Intent(this, TargetActivity.class);

// Intent to request shortcut creation
Intent intent = new Intent("com.android.launcher.action.INSTALL_SHORTCUT");
```

(continues on next page)

(continued from previous page)

```
// Specify an Intent to be launched when the shortcut is tapped
intent.putExtra(Intent.EXTRA_SHORTCUT_INTENT, targetIntent);
Parcelable icon =
    Intent.ShortcutIconResource.fromContext(context, iconResource);
intent.putExtra(Intent.EXTRA_SHORTCUT_ICON_RESOURCE, icon);
intent.putExtra(Intent.EXTRA_SHORTCUT_NAME, title);
intent.putExtra("duplicate", false);

// Use Broadcast to send the system our request for shortcut creation
context.sendBroadcast(intent);
```

In the Broadcast sent by the above code snippet, the receiver is the home-screen app, and it is difficult to identify the package name; one must take care to remember that this is a transmission to a public receiver with an implicit intent. Thus the Broadcast sent by this snippet could be received by any arbitrary app, including malware; for this reason, the inclusion of sensitive information in the Intent may create the risk of a damaging leak of information. It is particularly important to note that, when creating a URL-based shortcut, secret information may be contained in the URL itself.

As countermeasures, it is necessary to follow the points listed in "4.2.1.2. *Public Broadcast Receiver - Receiving/Sending Broadcasts*" and to ensure that the transmitted Intent does not contain sensitive information.

4.2.3.7 ACTION_CLOSE_SYSTEM_DIALOGS

ACTION_CLOSE_SYSTEM_DIALOGS is a Broadcast Intent that indicates the system dialog was closed. Furthermore, the system dialog can be closed by sending Broadcast from the application. The behavior of this ACTION_CLOSE_SYSTEM_DIALOGS varies by whether to target Android 12 or to target Android 11 or earlier as shown in the following.

If targeting Android 11 or earlier:

The following message is displayed on LogCat without executing the Intent.

```
E ActivityTaskManager Permission Denial: \
android.intent.action.CLOSE_SYSTEM_DIALOGS broadcast from \
com.package.name requires android.permission.BROADCAST_CLOSE_SYSTEM_DIALOGS, \
dropping broadcast.
```

However, in the following case, the system dialog can still be closed.

- Window displayed on the notification drawer
- The user operates the notification and the application processes services or Broadcast Receiver based on user actions
- The accessibility service is enabled

If targeting Android 12:

ACTION_CLOSE_SYSTEM_DIALOGS has been deprecated. SecurityException occurs if the application tries to invoke an Intent that includes this action.

However, in the following case, the system dialog can still be closed.

- If the application is running a single instrumentation test

If the accessibility service is enabled and it is required to close the notification bar, use the GLOBAL_ACTION_DISMISS_NOTIFICATION_SHADE accessibility action instead.

4.2.3.8 Enhanced Safety of Dynamic Broadcast Receiver

Until now, public broadcast receivers that were dynamic broadcast receivers were not allowed to configure export settings, but starting from Android 13, public/private can be specified. This is a specification that was added to enhance security, taking into account the risk of receiving broadcasts sent by malware.

To use this feature, set targetSdkVersion to 33 or higher, and perform the following procedure.

1. Enable DYNAMIC_RECEIVER_EXPLICIT_EXPORT_REQUIRED.
2. Specify RECEIVER_EXPORTED or RECEIVER_NOT_EXPORTED for the argument of registerReceiver.

```
context.registerReceiver(sharedBroadcastReceiver, intentFilter, RECEIVER_EXPORTED);
context.registerReceiver(sharedBroadcastReceiver, intentFilter, RECEIVER_NOT_
↪EXPORTED);
```

The DYNAMIC_RECEIVER_EXPLICIT_EXPORT_REQUIRED settings can be switched from the developer options or adb. When setting from the developer options, do so from the following screen.

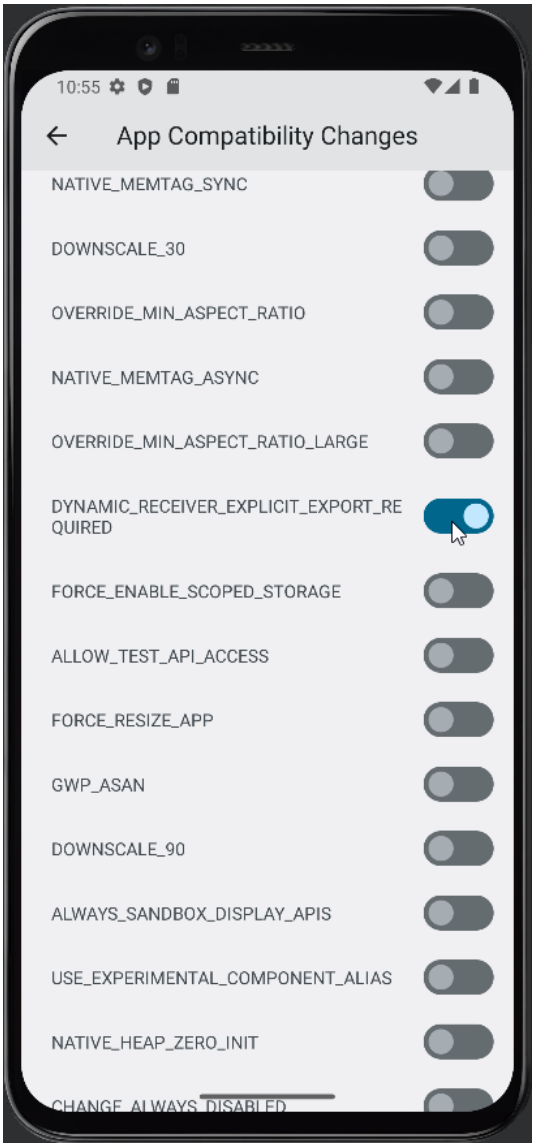


Fig. 4.2.6: DYNAMIC_RECEIVER_EXPLICIT_EXPORT_REQUIRED Setting(Developer Options)

If setting from adb, the following command lines appear.

```
$ adb shell am compat enable DYNAMIC_RECEIVER_EXPLICIT_EXPORT_REQUIRED org.jssec.
↪android.broadcast.publicreceiver
$ adb shell am compat disable DYNAMIC_RECEIVER_EXPLICIT_EXPORT_REQUIRED org.jssec.
↪android.broadcast.publicreceiver
```

If DYNAMIC_RECEIVER_EXPLICIT_EXPORT_REQUIRED was enabled, executing registerReceiver() without specifying RECEIVER_EXPORTED or RECEIVER_NOT_EXPORTED will trigger a SecurityException.

If DYNAMIC_RECEIVER_EXPLICIT_EXPORT_REQUIRED was disabled, when registerReceiver() is executed with RECEIVER_EXPORTED or RECEIVER_NOT_EXPORTED specified, the export setting is ignored, and the receiver operates as a conventional public broadcast receiver.

Note that starting with Android 14, DYNAMIC_RECEIVER_EXPLICIT_EXPORT_REQUIRED is enabled by default, and therefore, specifying of the export setting is mandatory for applications targeting Android 14.

However, for receivers that only receive system broadcasts[13], such as ACTION_TIME_TICK, the export setting is not necessary. At the time of writing, there are no errors or warnings at build or runtime even if the export setting is specified, but it is recommended that the export setting not be specified for such receivers in order to comply with future specifications.

- Example of registering a broadcast receiver that receives ACTION_TIME_TICK, which is a system broadcast

```
IntentFilter filter = new IntentFilter(Intent.ACTION_TIME_TICK);
receiver = new MyBroadcastReceiver();
registerReceiver(receiver, filter);
```

4.2.3.9 Changes to the Priority Specification for Ordered Broadcasts

Starting with Android 16, the specifications for specifying the priority of ordered broadcasts have changed significantly. Previously, Android allowed for detailed control of the reception order among multiple BroadcastReceivers by using IntentFilter#setPriority() or the android:priority attribute. For example, if there was a requirement to "always process a Receiver with a high priority value first," it was common to design something that relied on the priority value.

However, starting with Android 16, the way priority values are handled has changed fundamentally. No matter how high the priority value is, the system treats it as 0, making it effectively impossible to control the order by specifying a priority. Even if multiple Receivers are registered within an app, the order in which they are called is determined by the order of registration and internal system behavior, rather than the priority value. This means that order control that relies on priority values will completely break down.

For example, in the following code, ReceiverA with priority=1000 would always be called first on Android 15 and earlier, but on Android 16 and later, the order is no longer guaranteed.

```
val receiverA = object : BroadcastReceiver() {
    override fun onReceive(context: Context?, intent: Intent?) {
        Log.d("OrderedBroadcast", "ReceiverA (priority=1000) called")
    }
}
val filterA = IntentFilter(ACTION_TEST)
filterA.priority = 1000

val receiverB = object : BroadcastReceiver() {
    override fun onReceive(context: Context?, intent: Intent?) {
        Log.d("OrderedBroadcast", "ReceiverB (priority=0) called")
    }
}
val filterB = IntentFilter(ACTION_TEST)
```

(continues on next page)

[13] The list varies depending on the API Level, and in the case of 34, the following list is shown. /AppData/Local/Android/Sdk/platforms/android-34/data/broadcast_actions.txt

(continued from previous page)

```
filterB.priority = 0

registerReceiver(receiverA, filterA, RECEIVER_NOT_EXPORTED)
registerReceiver(receiverB, filterB, RECEIVER_NOT_EXPORTED)

sendOrderedBroadcast(Intent(ACTION_TEST).apply { `package` = packageName }, null)
```

Starting with Android 16, this code is no longer guaranteed to be called in the order ReceiverA → ReceiverB, and in some cases ReceiverB may be called first. Designs that rely on priority values carry a high risk of unexpected bugs or security breaches due to future OS updates or the addition or removal of Receivers.

In response to this specification change, developers should stop relying on priority value-dependent ordering and switch to a design that explicitly guarantees the order in a single function or class, rather than splitting processes that require a guaranteed order using BroadcastReceiver. For example, if you want to enforce processing in the order A -> B, rewrite it as follows:

```
private fun doOrderedProcess() {
    stepA()
    stepB()
}

private fun stepA() {
    Log.d("OrderedProcess", "Processing A")
    // Processing contents of A
}

private fun stepB() {
    Log.d("OrderedProcess", "Processing B")
    // Processing contents of B
}
```

By explicitly specifying the order with function calls in this way, processing will always occur in the intended order, regardless of the Android version or system behavior. If you want to separate the Receivers, you will need to revise your design to ensure the order yourself, such as by having the first Receiver call all the necessary processing in order.

Broadcasts are originally used when you want to notify multiple independent components or apps of the same event and are not suitable for processing where a guaranteed order is essential. If you need an order, the basis of secure development in Android 16 and later is to switch to a design that ensures the order yourself, using explicit calls, Services, callbacks, etc., rather than relying on Broadcast or priority.

4.2.3.10 Autonomous Re-pairing for Bluetooth Bond Loss

In Android 17 and later, "autonomous re-pairing," a system-level extension feature to automatically resolve Bluetooth connection disconnections (bond loss), has been introduced.

Conventionally, when binding with a peripheral device was lost, users themselves had to manually navigate to the settings screen, unpair, and perform pairing again. In Android 17, building upon the security enhancements of Android 16, the system can re-establish the connection in the background without requiring the user to manually navigate to settings, unpair the peripheral device, and re-pair.

Although code changes are not required for most apps, developers of dedicated apps operating in coordination with peripheral devices, such as smartwatches and IoT devices, or apps performing Bluetooth communication need to be aware of the following behavior changes in the Bluetooth stack:

- Changes in intent timing: The ACTION_KEY_MISSING intent is now broadcast only when an autonomous re-pairing attempt fails. This eliminates the need for apps to perform unnecessary error handling when the system successfully restores the bond in the background.

- New pairing context: The ACTION_PAIRING_REQUEST intent now includes the EXTRA_PAIRING_CONTEXT extra. This allows apps to distinguish between standard pairing requests and re-pairing attempts initiated autonomously by the system.
- Conditional key updates: Existing security keys are replaced only if re-pairing succeeds and the new connection meets or exceeds the security level of the previous bond. This functions as a protection to prevent security level downgrade attacks.
- User notifications: The system manages re-pairing via new UI notifications and dialogs. Messages asking users to confirm re-pairing attempts are displayed, allowing them to remain aware of reconnections.

Peripheral manufacturers and developers of integrated apps need to ensure that hardware and apps handle bond transitions appropriately. To test this behavior, it is recommended to simulate remote bond loss by manually deleting pairing information from the peripheral device or manually unpairing from Android's "Settings > Connected devices."

4.3 Creating/Using Content Providers

Since the interface of ContentResolver and SQLiteDatabase are so much alike, it's often misunderstood that Content Provider is so closely related to SQLiteDatabase. However, actually Content Provider simply provides the interface of inter-application data sharing, so it's necessary to pay attention that it does not interfere each data saving format. To save data in Content Provider, SQLiteDatabase can be used, and other saving formats, such as an XML file format, also can be used. Any data saving process is not included in the following sample code, so please add it if needed.

4.3.1 Sample Code

The risks and countermeasures of using Content Provider differ depending on how that Content Provider is being used. In this section, we have classified 5 types of Content Provider based on how the Content Provider is being used. You can find out which type of Content Provider you are supposed to create through the following chart shown below.

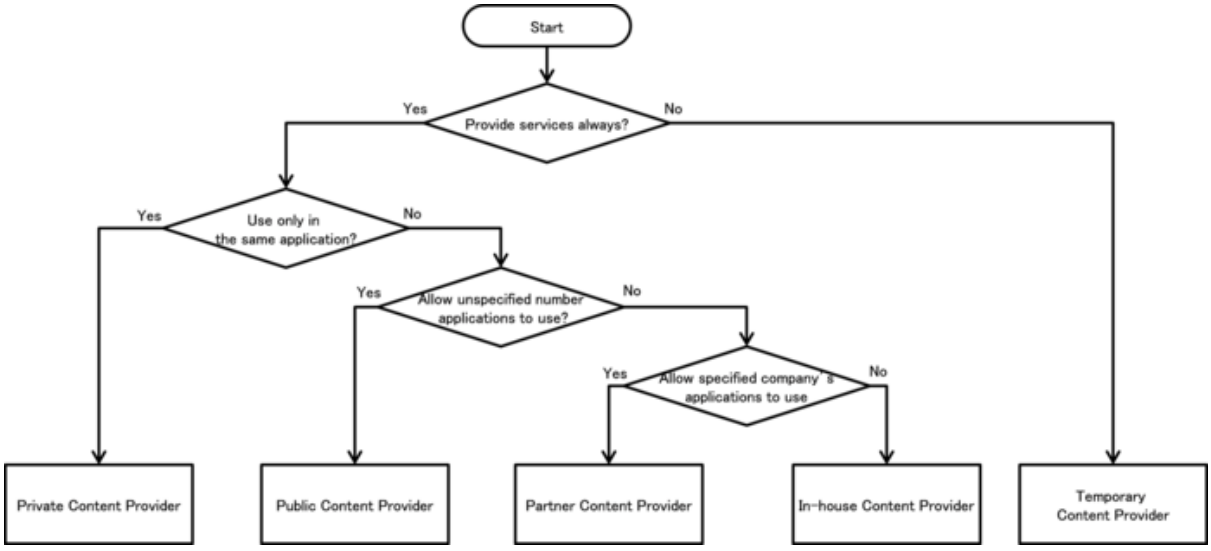


Fig. 4.3.1: Flow Figure to decide Content Provider Type

4.3.1.1 Creating/Using Private Content Providers

Private Content Provider is the Content Provider which is used only in the single application, and the safest Content Provider[14].

[14] However, non-public settings for Content Provider are not functional in Android 2.2 (API Level 8) and previous versions.

Sample code of how to implement a private Content Provider is shown below.

Points (Creating a Content Provider):

1. Explicitly set the exported attribute to false.
2. Handle the received request data carefully and securely, even though the data comes from the same application.
3. Sensitive information can be sent since it is sending and receiving all within the same application.

AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
          >

  <application
      android:icon="@drawable/ic_launcher"
      android:label="@string/app_name" >
    <activity
        android:name=".PrivateUserActivity"
        android:label="@string/app_name"
        android:exported="true" >
      <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER" />
      </intent-filter>
    </activity>

    <!-- *** POINT 1 *** Explicitly set the exported attribute to false. -->
    <provider
        android:name=".PrivateProvider"
        android:authorities="org.jssec.android.provider.privateprovider"
        android:exported="false" />
  </application>
</manifest>
```

PrivateProvider.java

```
/*
 * Copyright (C) 2012-2026 Japan Smartphone Security Association
 *
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 *
 *      http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */

package org.jssec.android.provider.privateprovider;

import android.content.ContentProvider;
import android.content.ContentUris;
import android.content.ContentValues;
import android.content.UriMatcher;
```

(continues on next page)

(continued from previous page)

```
import android.database.Cursor;
import android.database.MatrixCursor;
import android.net.Uri;

public class PrivateProvider extends ContentProvider {

    public static final String AUTHORITY =
            "org.jssec.android.provider.privateprovider";
    public static final String CONTENT_TYPE =
            "vnd.android.cursor.dir/vnd.org.jssec.contenttype";
    public static final String CONTENT_ITEM_TYPE =
            "vnd.android.cursor.item/vnd.org.jssec.contenttype";

    // Expose the interface that the Content Provider provides.
    public interface Download {
        public static final String PATH = "downloads";
        public static final Uri CONTENT_URI =
                Uri.parse("content://" + AUTHORITY + "/" + PATH);
    }
    public interface Address {
        public static final String PATH = "addresses";
        public static final Uri CONTENT_URI =
                Uri.parse("content://" + AUTHORITY + "/" + PATH);
    }

    // UriMatcher
    private static final int DOWNLOADS_CODE = 1;
    private static final int DOWNLOADS_ID_CODE = 2;
    private static final int ADDRESSES_CODE = 3;
    private static final int ADDRESSES_ID_CODE = 4;
    private static UriMatcher sUriMatcher;
    static {
        sUriMatcher = new UriMatcher(UriMatcher.NO_MATCH);
        sUriMatcher.addURI(AUTHORITY, Download.PATH, DOWNLOADS_CODE);
        sUriMatcher.addURI(AUTHORITY, Download.PATH + "/#", DOWNLOADS_ID_CODE);
        sUriMatcher.addURI(AUTHORITY, Address.PATH, ADDRESSES_CODE);
        sUriMatcher.addURI(AUTHORITY, Address.PATH + "/#", ADDRESSES_ID_CODE);
    }

    // Since this is a sample program, query method returns the following
    // fixed result always without using database.
    private static MatrixCursor sAddressCursor =
            new MatrixCursor(new String[] { "_id", "city" });
    static {
        sAddressCursor.addRow(new String[] { "1", "New York" });
        sAddressCursor.addRow(new String[] { "2", "Longon" });
        sAddressCursor.addRow(new String[] { "3", "Paris" });
    }
    private static MatrixCursor sDownloadCursor =
            new MatrixCursor(new String[] { "_id", "path" });
    static {
        sDownloadCursor.addRow(new String[] { "1", "/sdcard/downloads/sample.jpg" }
↪);
        sDownloadCursor.addRow(new String[] { "2", "/sdcard/downloads/sample.txt" }
↪);
    }
```

(continues on next page)

(continued from previous page)

```
    @Override
    public boolean onCreate() {
        return true;
    }

    @Override
    public String getType(Uri uri) {
        // *** POINT 2 *** Handle the received request data carefully and securely,
        // even though the data comes from the same application.
        // Here, whether uri is within expectations or not, is verified by
        // UriMatcher#match() and switch case.
        // Checking for other parameters are omitted here, due to sample.
        // Please refer to "3.2 Handle Input Data Carefully and Securely."

        // *** POINT 3 *** Sensitive information can be sent since it is sending
        // and receiving all within the same application.
        // However, the result of getType rarely has the sensitive meaning.
        switch (sUriMatcher.match(uri)) {
        case DOWNLOADS_CODE:
        case ADDRESSES_CODE:
            return CONTENT_TYPE;

        case DOWNLOADS_ID_CODE:
        case ADDRESSES_ID_CODE:
            return CONTENT_ITEM_TYPE;

        default:
            throw new IllegalArgumentException("Invalid URI:" + uri);
        }
    }

    @Override
    public Cursor query(Uri uri, String[] projection, String selection,
            String[] selectionArgs, String sortOrder) {

        // *** POINT 2 *** Handle the received request data carefully and securely,
        // even though the data comes from the same application.
        // Here, whether uri is within expectations or not, is verified by
        // UriMatcher#match() and switch case.
        // Checking for other parameters are omitted here, due to sample.
        // Please refer to "3.2 Handle Input Data Carefully and Securely."

        // *** POINT 3 *** Sensitive information can be sent since it is sending
        // and receiving all within the same application.
        // It depends on application whether the query result has sensitive meaning
        // or not.
        switch (sUriMatcher.match(uri)) {
        case DOWNLOADS_CODE:
        case DOWNLOADS_ID_CODE:
            return sDownloadCursor;

        case ADDRESSES_CODE:
        case ADDRESSES_ID_CODE:
            return sAddressCursor;
```

(continues on next page)

(continued from previous page)

```
        default:
            throw new IllegalArgumentException("Invalid URI:" + uri);
        }
    }

    @Override
    public Uri insert(Uri uri, ContentValues values) {

        // *** POINT 2 *** Handle the received request data carefully and securely,
        // even though the data comes from the same application.
        // Here, whether uri is within expectations or not, is verified by
        // UriMatcher#match() and switch case.
        // Checking for other parameters are omitted here, due to sample.
        // Please refer to "3.2 Handle Input Data Carefully and Securely."

        // *** POINT 3 *** Sensitive information can be sent since it is sending
        // and receiving all within the same application.
        // It depends on application whether the issued ID has sensitive meaning
        // or not.
        switch (sUriMatcher.match(uri)) {
        case DOWNLOADS_CODE:
            return ContentUris.withAppendedId(Download.CONTENT_URI, 3);

        case ADDRESSES_CODE:
            return ContentUris.withAppendedId(Address.CONTENT_URI, 4);

        default:
            throw new IllegalArgumentException("Invalid URI:" + uri);
        }
    }

    @Override
    public int update(Uri uri, ContentValues values, String selection,
            String[] selectionArgs) {

        // *** POINT 2 *** Handle the received request data carefully and securely,
        // even though the data comes from the same application.
        // Here, whether uri is within expectations or not, is verified by
        // UriMatcher#match() and switch case.
        // Checking for other parameters are omitted here, due to sample.
        // Please refer to "3.2 Handle Input Data Carefully and Securely."

        // *** POINT 3 *** Sensitive information can be sent since it is sending
        // and receiving all within the same application.
        // It depends on application whether the number of updated records has
        // sensitive meaning or not.
        switch (sUriMatcher.match(uri)) {
        case DOWNLOADS_CODE:
                return 5;       // Return number of updated records

        case DOWNLOADS_ID_CODE:
            return 1;

        case ADDRESSES_CODE:
                return 15;
```

(continues on next page)

(continued from previous page)

```
        case ADDRESSES_ID_CODE:
            return 1;

        default:
            throw new IllegalArgumentException("Invalid URI:" + uri);
        }
    }

    @Override
    public int delete(Uri uri, String selection, String[] selectionArgs) {

        // *** POINT 2 *** Handle the received request data carefully and securely,
        // even though the data comes from the same application.
        // Here, whether uri is within expectations or not, is verified by
        // UriMatcher#match() and switch case.
        // Checking for other parameters are omitted here, due to sample.
        // Please refer to "3.2 Handle Input Data Carefully and Securely."

        // *** POINT 3 *** Sensitive information can be sent since it is sending
        // and receiving all within the same application.
        // It depends on application whether the number of deleted records has
        // sensitive meaning or not.
        switch (sUriMatcher.match(uri)) {
        case DOWNLOADS_CODE:
                return 10;      // Return number of deleted records

        case DOWNLOADS_ID_CODE:
            return 1;

        case ADDRESSES_CODE:
                return 20;

        case ADDRESSES_ID_CODE:
            return 1;

        default:
            throw new IllegalArgumentException("Invalid URI:" + uri);
        }
    }
}
```

Next is an example of Activity which uses Private Content Provider.

Points (Using a Content Provider):

4. Sensitive information can be sent since the destination provider is in the same application.
5. Handle received result data carefully and securely, even though the data comes from the same application.

PrivateUserActivity.java

```
/*
 * Copyright (C) 2012-2026 Japan Smartphone Security Association
 *
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 *
 *      http://www.apache.org/licenses/LICENSE-2.0
```

(continues on next page)

(continued from previous page)

```
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */

package org.jssec.android.provider.privateprovider;

import android.app.Activity;
import android.database.Cursor;
import android.net.Uri;
import android.os.Bundle;
import android.view.View;
import android.widget.TextView;

public class PrivateUserActivity extends Activity {

    public void onQueryClick(View view) {

        logLine("[Query]");

        Cursor cursor = null;
        try {
            // *** POINT 4 *** Sensitive information can be sent since the
            // destination provider is in the same application.
            cursor =
                getContentResolver().query(PrivateProvider.Download.CONTENT_URI,
                                                        null, null, null, null);

            // *** POINT 5 *** Handle received result data carefully and securely,
            // even though the data comes from the same application.
            // Omitted, since this is a sample. Please refer to
            // "3.2 Handling Input Data Carefully and Securely."
            if (cursor == null) {
                logLine("  null cursor");
            } else {
                boolean moved = cursor.moveToFirst();
                while (moved) {
                    logLine(String.format("  %d, %s", cursor.getInt(0),
                                            cursor.getString(1)));
                    moved = cursor.moveToNext();
                }
            }
        }
        finally {
            if (cursor != null) cursor.close();
        }
    }

    public void onInsertClick(View view) {

        logLine("[Insert]");

        // *** POINT 4 *** Sensitive information can be sent since the
```

(continues on next page)

(continued from previous page)

```
        // destination provider is in the same application.
        Uri uri =
            getContentResolver().insert(PrivateProvider.Download.CONTENT_URI,
                                        null);

        // *** POINT 5 *** Handle received result data carefully and securely,
        // even though the data comes from the same application.
        // Omitted, since this is a sample. Please refer to
        // "3.2 Handling Input Data Carefully and Securely."
        logLine("  uri:" + uri);
    }

    public void onUpdateClick(View view) {

        logLine("[Update]");

        // *** POINT 4 *** Sensitive information can be sent since the
        // destination provider is in the same application.
        int count =
            getContentResolver().update(PrivateProvider.Download.CONTENT_URI,
                                        null, null, null);

        // *** POINT 5 *** Handle received result data carefully and securely,
        // even though the data comes from the same application.
        // Omitted, since this is a sample. Please refer to
        // "3.2 Handling Input Data Carefully and Securely."
        logLine(String.format("  %s records updated", count));
    }

    public void onDeleteClick(View view) {

        logLine("[Delete]");

        // *** POINT 4 *** Sensitive information can be sent since the
        // destination provider is in the same application.
        int count =
            getContentResolver().delete(PrivateProvider.Download.CONTENT_URI,
                                        null, null);

        // *** POINT 5 *** Handle received result data carefully and securely,
        // even though the data comes from the same application.
        // Omitted, since this is a sample. Please refer to
        // "3.2 Handling Input Data Carefully and Securely."
        logLine(String.format("  %s records deleted", count));
    }

    private TextView mLogView;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        mLogView = (TextView)findViewById(R.id.logview);
    }

    private void logLine(String line) {
```

(continues on next page)

(continued from previous page)

```
        mLogView.append(line);
        mLogView.append("\n");
    }
}
```

4.3.1.2 Creating/Using Public Content Providers

Public Content Provider is the Content Provider which is supposed to be used by unspecified large number of applications. It's necessary to pay attention that since this doesn't specify clients, it may be attacked and tampered by Malware. For example, a saved data may be taken by select(), a data may be changed by update(), or a fake data may be inserted/deleted by insert()/delete().

In addition, when using a custom Public Content Provider which is not provided by Android OS, it's necessary to pay attention that request parameter may be received by Malware which masquerades as the custom Public Content Provider, and also the attack result data may be sent. Contacts and MediaStore provided by Android OS are also Public Content Providers, but Malware cannot masquerades as them.

Sample code to implement a Public Content Provider is shown below.

Points (Creating a Content Provider):

1. Explicitly set the exported attribute to true.
2. Handle the received request data carefully and securely.
3. When returning a result, do not include sensitive information.

AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
          >

  <application
      android:icon="@drawable/ic_launcher"
      android:label="@string/app_name" >

    <!-- *** POINT 1 *** Explicitly set the exported attribute to true. -->
    <provider
        android:name=".PublicProvider"
        android:authorities="org.jssec.android.provider.publicprovider"
        android:exported="true" />
  </application>
</manifest>
```

PublicProvider.java

```
/*
 * Copyright (C) 2012-2026 Japan Smartphone Security Association
 *
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 *
 *      http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
```

(continues on next page)

(continued from previous page)

```
 * limitations under the License.
 */

package org.jssec.android.provider.publicprovider;

import android.content.ContentProvider;
import android.content.ContentUris;
import android.content.ContentValues;
import android.content.UriMatcher;
import android.database.Cursor;
import android.database.MatrixCursor;
import android.net.Uri;

public class PublicProvider extends ContentProvider {

    public static final String AUTHORITY =
            "org.jssec.android.provider.publicprovider";
    public static final String CONTENT_TYPE =
            "vnd.android.cursor.dir/vnd.org.jssec.contenttype";
    public static final String CONTENT_ITEM_TYPE =
            "vnd.android.cursor.item/vnd.org.jssec.contenttype";

    // Expose the interface that the Content Provider provides.
    public interface Download {
        public static final String PATH = "downloads";
        public static final Uri CONTENT_URI =
                Uri.parse("content://" + AUTHORITY + "/" + PATH);
    }
    public interface Address {
        public static final String PATH = "addresses";
        public static final Uri CONTENT_URI =
                Uri.parse("content://" + AUTHORITY + "/" + PATH);
    }

    // UriMatcher
    private static final int DOWNLOADS_CODE = 1;
    private static final int DOWNLOADS_ID_CODE = 2;
    private static final int ADDRESSES_CODE = 3;
    private static final int ADDRESSES_ID_CODE = 4;
    private static UriMatcher sUriMatcher;
    static {
        sUriMatcher = new UriMatcher(UriMatcher.NO_MATCH);
        sUriMatcher.addURI(AUTHORITY, Download.PATH, DOWNLOADS_CODE);
        sUriMatcher.addURI(AUTHORITY, Download.PATH + "/#", DOWNLOADS_ID_CODE);
        sUriMatcher.addURI(AUTHORITY, Address.PATH, ADDRESSES_CODE);
        sUriMatcher.addURI(AUTHORITY, Address.PATH + "/#", ADDRESSES_ID_CODE);
    }

    // Since this is a sample program,
    // query method returns the following fixed result always without using
    // database.
    private static MatrixCursor sAddressCursor =
            new MatrixCursor(new String[] { "_id", "city" });
    static {
        sAddressCursor.addRow(new String[] { "1", "New York" });
        sAddressCursor.addRow(new String[] { "2", "London" });
```

(continues on next page)

(continued from previous page)

```
        sAddressCursor.addRow(new String[] { "3", "Paris" });
    }
    private static MatrixCursor sDownloadCursor =
            new MatrixCursor(new String[] { "_id", "path" });
    static {
        sDownloadCursor.addRow(new String[] { "1", "/sdcard/downloads/sample.jpg" }
↪);
        sDownloadCursor.addRow(new String[] { "2", "/sdcard/downloads/sample.txt" }
↪);
    }

    @Override
    public boolean onCreate() {
        return true;
    }

    @Override
    public String getType(Uri uri) {

        switch (sUriMatcher.match(uri)) {
        case DOWNLOADS_CODE:
        case ADDRESSES_CODE:
            return CONTENT_TYPE;

        case DOWNLOADS_ID_CODE:
        case ADDRESSES_ID_CODE:
            return CONTENT_ITEM_TYPE;

        default:
            throw new IllegalArgumentException("Invalid URI:" + uri);
        }
    }

    @Override
    public Cursor query(Uri uri, String[] projection, String selection,
                        String[] selectionArgs, String sortOrder) {

        // *** POINT 2 *** Handle the received request data carefully and securely.
        // Here, whether uri is within expectations or not, is verified by
        // UriMatcher#match() and switch case.
        // Checking for other parameters are omitted here, due to sample.
        // Refer to "3.2 Handle Input Data Carefully and Securely."

        // *** POINT 3 *** When returning a result, do not include sensitive
        // information.
        // It depends on application whether the query result has sensitive
        // meaning or not.
        // If no problem when the information is taken by malware, it can be
        // returned as result.
        switch (sUriMatcher.match(uri)) {
        case DOWNLOADS_CODE:
        case DOWNLOADS_ID_CODE:
            return sDownloadCursor;

        case ADDRESSES_CODE:
        case ADDRESSES_ID_CODE:
```

(continues on next page)

(continued from previous page)

```
            return sAddressCursor;

        default:
            throw new IllegalArgumentException("Invalid URI:" + uri);
        }
    }

    @Override
    public Uri insert(Uri uri, ContentValues values) {

        // *** POINT 2 *** Handle the received request data carefully and securely.
        // Here, whether uri is within expectations or not, is verified by
        // UriMatcher#match() and switch case.
        // Checking for other parameters are omitted here, due to sample.
        // Refer to "3.2 Handle Input Data Carefully and Securely."

        // *** POINT 3 *** When returning a result, do not include sensitive
        // information.
        // It depends on application whether the issued ID has sensitive
        // meaning or not.
        // If no problem when the information is taken by malware, it can be
        // returned as result.
        switch (sUriMatcher.match(uri)) {
        case DOWNLOADS_CODE:
            return ContentUris.withAppendedId(Download.CONTENT_URI, 3);

        case ADDRESSES_CODE:
            return ContentUris.withAppendedId(Address.CONTENT_URI, 4);

        default:
            throw new IllegalArgumentException("Invalid URI:" + uri);
        }
    }

    @Override
    public int update(Uri uri, ContentValues values, String selection,
                      String[] selectionArgs) {

        // *** POINT 2 *** Handle the received request data carefully and securely.
        // Here, whether uri is within expectations or not, is verified by
        // UriMatcher#match() and switch case.
        // Checking for other parameters are omitted here, due to sample.
        // Refer to "3.2 Handle Input Data Carefully and Securely."

        // *** POINT 3 *** When returning a result, do not include sensitive
        // information.
        // It depends on application whether the number of updated records has
        // sensitive meaning or not.
        // If no problem when the information is taken by malware, it can be
        // returned as result.
        switch (sUriMatcher.match(uri)) {
        case DOWNLOADS_CODE:
            return 5;   // Return number of updated records

        case DOWNLOADS_ID_CODE:
            return 1;
```

(continues on next page)

(continued from previous page)

```
        case ADDRESSES_CODE:
            return 15;

        case ADDRESSES_ID_CODE:
            return 1;

        default:
            throw new IllegalArgumentException("Invalid URI:" + uri);
        }
    }

    @Override
    public int delete(Uri uri, String selection, String[] selectionArgs) {

        // *** POINT 2 *** Handle the received request data carefully and securely.
        // Here, whether uri is within expectations or not, is verified by
        // UriMatcher#match() and switch case.
        // Checking for other parameters are omitted here, due to sample.
        // Refer to "3.2 Handle Input Data Carefully and Securely."

        // *** POINT 3 *** When returning a result, do not include sensitive
        // information.
        // It depends on application whether the number of deleted records has
        // sensitive meaning or not.
        // If no problem when the information is taken by malware, it can be
        // returned as result.
        switch (sUriMatcher.match(uri)) {
        case DOWNLOADS_CODE:
            return 10;  // Return number of deleted records

        case DOWNLOADS_ID_CODE:
            return 1;

        case ADDRESSES_CODE:
            return 20;

        case ADDRESSES_ID_CODE:
            return 1;

        default:
            throw new IllegalArgumentException("Invalid URI:" + uri);
        }
    }
}
```

Next is an example of Activity which uses Public Content Provider.

Points (Using a Content Provider):

4. Do not send sensitive information.
5. When receiving a result, handle the result data carefully and securely.

PublicUserActivity.java

```
/*
 * Copyright (C) 2012-2026 Japan Smartphone Security Association
 *
```

(continues on next page)

(continued from previous page)

```
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 *
 *      http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */

package org.jssec.android.provider.publicuser;

import android.app.Activity;
import android.content.ContentValues;
import android.content.pm.ProviderInfo;
import android.database.Cursor;
import android.net.Uri;
import android.os.Bundle;
import android.view.View;
import android.widget.TextView;

public class PublicUserActivity extends Activity {

    // Target Content Provider Information
    private static final String AUTHORITY =
            "org.jssec.android.provider.publicprovider";
    private interface Address {
        public static final String PATH = "addresses";
        public static final Uri CONTENT_URI =
                Uri.parse("content://" + AUTHORITY + "/" + PATH);
    }

    public void onQueryClick(View view) {

        logLine("[Query]");

        if (!providerExists(Address.CONTENT_URI)) {
            logLine("  Content Provider doesn't exist.");
            return;
        }

        Cursor cursor = null;
        try {
            // *** POINT 4 *** Do not send sensitive information.
            // since the target Content Provider may be malware.
            // If no problem when the information is taken by malware,
            // it can be included in the request.
            cursor = getContentResolver().query(Address.CONTENT_URI,
                                                null, null, null, null);

            // *** POINT 5 *** When receiving a result, handle the result data
            // carefully and securely.
            // Omitted, since this is a sample. Please refer to
```

(continues on next page)

(continued from previous page)

```
            // "3.2 Handling Input Data Carefully and Securely."
            if (cursor == null) {
                logLine("  null cursor");
            } else {
                boolean moved = cursor.moveToFirst();
                while (moved) {
                    logLine(String.format("  %d, %s", cursor.getInt(0),
                                          cursor.getString(1)));
                    moved = cursor.moveToNext();
                }
            }
        }
        finally {
            if (cursor != null) cursor.close();
        }
    }

    public void onInsertClick(View view) {

        logLine("[Insert]");

        if (!providerExists(Address.CONTENT_URI)) {
            logLine("  Content Provider doesn't exist.");
            return;
        }

        // *** POINT 4 *** Do not send sensitive information.
        // since the target Content Provider may be malware.
        // If no problem when the information is taken by malware,
        // it can be included in the request.
        ContentValues values = new ContentValues();
        values.put("city", "Tokyo");
        Uri uri = getContentResolver().insert(Address.CONTENT_URI, values);

        // *** POINT 5 *** When receiving a result, handle the result data
        // carefully and securely.
        // Omitted, since this is a sample. Please refer to
        // "3.2 Handling Input Data Carefully and Securely."
        logLine("  uri:" + uri);
    }

    public void onUpdateClick(View view) {

        logLine("[Update]");

        if (!providerExists(Address.CONTENT_URI)) {
            logLine("  Content Provider doesn't exist.");
            return;
        }

        // *** POINT 4 *** Do not send sensitive information.
        // since the target Content Provider may be malware.
        // If no problem when the information is taken by malware,
        // it can be included in the request.
        ContentValues values = new ContentValues();
        values.put("city", "Tokyo");
```

(continues on next page)

(continued from previous page)

```
        String where = "_id = ?";
        String[] args = { "4" };
        int count =
            getContentResolver().update(Address.CONTENT_URI, values, where, args);

        // *** POINT 5 *** When receiving a result, handle the result data
        // carefully and securely.
        // Omitted, since this is a sample. Please refer to
        // "3.2 Handling Input Data Carefully and Securely."
        logLine(String.format("  %s records updated", count));
    }

    public void onDeleteClick(View view) {

        logLine("[Delete]");

        if (!providerExists(Address.CONTENT_URI)) {
            logLine("  Content Provider doesn't exist.");
            return;
        }

        // *** POINT 4 *** Do not send sensitive information.
        // since the target Content Provider may be malware.
        // If no problem when the information is taken by malware,
        // it can be included in the request.
        int count = getContentResolver().delete(Address.CONTENT_URI, null, null);

        // *** POINT 5 *** When receiving a result, handle the result data
        // carefully and securely.
        // Omitted, since this is a sample. Please refer to
        // "3.2 Handling Input Data Carefully and Securely."
        logLine(String.format("  %s records deleted", count));
    }

    private boolean providerExists(Uri uri) {
        ProviderInfo pi =
            getPackageManager().resolveContentProvider(uri.getAuthority(), 0);
        return (pi != null);
    }

    private TextView mLogView;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        mLogView = (TextView)findViewById(R.id.logview);
    }

    private void logLine(String line) {
        mLogView.append(line);
        mLogView.append("\n");
    }
}
```

4.3.1.3 Creating/Using Partner Content Providers

Partner Content Provider is the Content Provider which can be used only by the particular applications. The system consists of a partner company's application and In-house application, and it is used to protect the information and features which are handled between a partner application and an In-house application.

Sample code to implement a partner-only Content Provider is shown below.

Points (Creating a Content Provider):

1. Explicitly set the exported attribute to true.
2. Verify if the certificate of a requesting application has been registered in the own white list.
3. Handle the received request data carefully and securely, even though the data comes from a partner application.
4. Information that is granted to disclose to partner applications can be returned.

AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
          >

  <application
      android:icon="@drawable/ic_launcher"
      android:label="@string/app_name" >

    <!-- *** POINT 1 *** Explicitly set the exported attribute to true. -->
    <provider
        android:name=".PartnerProvider"
        android:authorities="org.jssec.android.provider.partnerprovider"
        android:exported="true" />
  </application>
</manifest>
```

PartnerProvider.java

```
/*
 * Copyright (C) 2012-2026 Japan Smartphone Security Association
 *
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 *
 *      http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */

package org.jssec.android.provider.partnerprovider;

import java.util.List;

import org.jssec.android.shared.PkgCertWhitelists;
import org.jssec.android.shared.Utils;

import android.app.ActivityManager;
```

(continues on next page)

(continued from previous page)

```
import android.app.ActivityManager.RunningAppProcessInfo;
import android.content.ContentProvider;
import android.content.ContentUris;
import android.content.ContentValues;
import android.content.Context;
import android.content.UriMatcher;
import android.database.Cursor;
import android.database.MatrixCursor;
import android.net.Uri;
import android.os.Binder;
import android.os.Build;

public class PartnerProvider extends ContentProvider {

    public static final String AUTHORITY =
            "org.jssec.android.provider.partnerprovider";
    public static final String CONTENT_TYPE =
            "vnd.android.cursor.dir/vnd.org.jssec.contenttype";
    public static final String CONTENT_ITEM_TYPE =
            "vnd.android.cursor.item/vnd.org.jssec.contenttype";

    // Expose the interface that the Content Provider provides.
    public interface Download {
        public static final String PATH = "downloads";
        public static final Uri CONTENT_URI =
                Uri.parse("content://" + AUTHORITY + "/" + PATH);
    }
    public interface Address {
        public static final String PATH = "addresses";
        public static final Uri CONTENT_URI =
                Uri.parse("content://" + AUTHORITY + "/" + PATH);
    }

    // UriMatcher
    private static final int DOWNLOADS_CODE = 1;
    private static final int DOWNLOADS_ID_CODE = 2;
    private static final int ADDRESSES_CODE = 3;
    private static final int ADDRESSES_ID_CODE = 4;
    private static UriMatcher sUriMatcher;
    static {
        sUriMatcher = new UriMatcher(UriMatcher.NO_MATCH);
        sUriMatcher.addURI(AUTHORITY, Download.PATH, DOWNLOADS_CODE);
        sUriMatcher.addURI(AUTHORITY, Download.PATH + "/#", DOWNLOADS_ID_CODE);
        sUriMatcher.addURI(AUTHORITY, Address.PATH, ADDRESSES_CODE);
        sUriMatcher.addURI(AUTHORITY, Address.PATH + "/#", ADDRESSES_ID_CODE);
    }

    // Since this is a sample program,
    // query method returns the following fixed result always without using
    // database.
    private static MatrixCursor sAddressCursor =
            new MatrixCursor(new String[] { "_id", "city" });
    static {
        sAddressCursor.addRow(new String[] { "1", "New York" });
        sAddressCursor.addRow(new String[] { "2", "London" });
        sAddressCursor.addRow(new String[] { "3", "Paris" });
```

(continues on next page)

(continued from previous page)

```
    }
    private static MatrixCursor sDownloadCursor =
            new MatrixCursor(new String[] { "_id", "path" });
    static {
        sDownloadCursor.addRow(new String[] { "1", "/sdcard/downloads/sample.jpg" }
↪);
        sDownloadCursor.addRow(new String[] { "2", "/sdcard/downloads/sample.txt" }
↪);
    }

    // *** POINT 2 *** Verify if the certificate of a requesting application has
    // been registered in the own white list.
    private static PkgCertWhitelists sWhitelists = null;
    private static void buildWhitelists(Context context) {
        boolean isdebug = Utils.isDebuggable(context);
        sWhitelists = new PkgCertWhitelists();

        // Register certificate hash value of partner application
        // org.jssec.android.provider.partneruser.
        sWhitelists.add("org.jssec.android.provider.partneruser", isdebug ?
            // Certificate hash value of "androiddebugkey" in the debug.keystore.
            "0EFB7236 328348A9 89718BAD DF57F544 D5CCB4AE B9DB34BC 1E29DD26␣
↪F77C8255" :
            // Certificate hash value of "partner key" in the keystore.
            "1F039BB5 7861C27A 3916C778 8E78CE00 690B3974 3EB8259F E2627B8D␣
↪4C0EC35A");

        // Register following other partner applications in the same way.
    }
    private static boolean checkPartner(Context context, String pkgname) {
        if (sWhitelists == null) buildWhitelists(context);
        return sWhitelists.test(context, pkgname);
    }
    // Get the package name of the calling application.
    private String getCallingPackage(Context context) {
        String pkgname;
        if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.KITKAT) {
            pkgname = super.getCallingPackage();
        } else {
            pkgname = null;
            ActivityManager am = (ActivityManager) context.
↪getSystemService(Context.ACTIVITY_SERVICE);
            List<RunningAppProcessInfo> procList = am.getRunningAppProcesses();
            int callingPid = Binder.getCallingPid();
            if (procList != null) {
                for (RunningAppProcessInfo proc : procList) {
                    if (proc.pid == callingPid) {
                        pkgname = proc.pkgList[proc.pkgList.length - 1];
                        break;
                    }
                }
            }
        }
        return pkgname;
    }
```

(continues on next page)

(continued from previous page)

```
    @Override
    public boolean onCreate() {
        return true;
    }

    @Override
    public String getType(Uri uri) {

        switch (sUriMatcher.match(uri)) {
        case DOWNLOADS_CODE:
        case ADDRESSES_CODE:
            return CONTENT_TYPE;

        case DOWNLOADS_ID_CODE:
        case ADDRESSES_ID_CODE:
            return CONTENT_ITEM_TYPE;

        default:
            throw new IllegalArgumentException("Invalid URI:" + uri);
        }
    }

    @Override
    public Cursor query(Uri uri, String[] projection, String selection,
                        String[] selectionArgs, String sortOrder) {

        // *** POINT 2 *** Verify if the certificate of a requesting application
        // has been registered in the own white list.
        if (!checkPartner(getContext(), getCallingPackage(getContext()))) {
            throw new SecurityException("Calling application is not a partner␣
↪application.");
        }

        // *** POINT 3 *** Handle the received request data carefully and securely,
        // even though the data comes from a partner application.
        // Here, whether uri is within expectations or not, is verified by
        // UriMatcher#match() and switch case.
        // Checking for other parameters are omitted here, due to sample.
        // Refer to "3.2 Handle Input Data Carefully and Securely."

        // *** POINT 4 *** Information that is granted to disclose to partner
        // applications can be returned.
        // It depends on application whether the query result can be disclosed
        // or not.
        switch (sUriMatcher.match(uri)) {
        case DOWNLOADS_CODE:
        case DOWNLOADS_ID_CODE:
            return sDownloadCursor;

        case ADDRESSES_CODE:
        case ADDRESSES_ID_CODE:
            return sAddressCursor;

        default:
            throw new IllegalArgumentException("Invalid URI:" + uri);
        }
```

(continues on next page)

(continued from previous page)

```
    }

    @Override
    public Uri insert(Uri uri, ContentValues values) {

        // *** POINT 2 *** Verify if the certificate of a requesting application
        // has been registered in the own white list.
        if (!checkPartner(getContext(), getCallingPackage(getContext()))) {
            throw new SecurityException("Calling application is not a partner␣
↪application.");
        }

        // *** POINT 3 *** Handle the received request data carefully and securely,
        // even though the data comes from a partner application.
        // Here, whether uri is within expectations or not, is verified by
        // UriMatcher#match() and switch case.
        // Checking for other parameters are omitted here, due to sample.
        // Refer to "3.2 Handle Input Data Carefully and Securely."

        // *** POINT 4 *** Information that is granted to disclose to partner
        // applications can be returned.
        // It depends on application whether the issued ID has sensitive meaning
        // or not.
        switch (sUriMatcher.match(uri)) {
        case DOWNLOADS_CODE:
            return ContentUris.withAppendedId(Download.CONTENT_URI, 3);

        case ADDRESSES_CODE:
            return ContentUris.withAppendedId(Address.CONTENT_URI, 4);

        default:
            throw new IllegalArgumentException("Invalid URI:" + uri);
        }
    }

    @Override
    public int update(Uri uri, ContentValues values, String selection,
                      String[] selectionArgs) {

        // *** POINT 2 *** Verify if the certificate of a requesting application
        // has been registered in the own white list.
        if (!checkPartner(getContext(), getCallingPackage(getContext()))) {
            throw new SecurityException("Calling application is not a partner␣
↪application.");
        }

        // *** POINT 3 *** Handle the received request data carefully and securely,
        // even though the data comes from a partner application.
        // Here, whether uri is within expectations or not, is verified by
        // UriMatcher#match() and switch case.
        // Checking for other parameters are omitted here, due to sample.
        // Refer to "3.2 Handle Input Data Carefully and Securely."

        // *** POINT 4 *** Information that is granted to disclose to partner
        // applications can be returned.
        // It depends on application whether the number of updated records has
```

(continues on next page)

(continued from previous page)

```
        // sensitive meaning or not.
        switch (sUriMatcher.match(uri)) {
        case DOWNLOADS_CODE:
            return 5;   // Return number of updated records

        case DOWNLOADS_ID_CODE:
            return 1;

        case ADDRESSES_CODE:
            return 15;

        case ADDRESSES_ID_CODE:
            return 1;

        default:
            throw new IllegalArgumentException("Invalid URI:" + uri);
        }
    }

    @Override
    public int delete(Uri uri, String selection, String[] selectionArgs) {

        // *** POINT 2 *** Verify if the certificate of a requesting application
        // has been registered in the own white list.
        if (!checkPartner(getContext(), getCallingPackage(getContext()))) {
            throw new SecurityException("Calling application is not a partner␣
↪application.");
        }

        // *** POINT 3 *** Handle the received request data carefully and securely,
        // even though the data comes from a partner application.
        // Here, whether uri is within expectations or not, is verified by
        // UriMatcher#match() and switch case.
        // Checking for other parameters are omitted here, due to sample.
        // Refer to "3.2 Handle Input Data Carefully and Securely."

        // *** POINT 4 *** Information that is granted to disclose to partner
        // applications can be returned.
        // It depends on application whether the number of deleted records has
        // sensitive meaning or not.
        switch (sUriMatcher.match(uri)) {
        case DOWNLOADS_CODE:
            return 10;  // Return number of deleted records

        case DOWNLOADS_ID_CODE:
            return 1;

        case ADDRESSES_CODE:
            return 20;

        case ADDRESSES_ID_CODE:
            return 1;

        default:
            throw new IllegalArgumentException("Invalid URI:" + uri);
        }
```

(continues on next page)

(continued from previous page)

```
    }
}
```

Next is an example of Activity which use partner only Content Provider.

Points (Using a Content Provider):

5. Verify if the certificate of the target application has been registered in the own white list.
6. Information that is granted to disclose to partner applications can be sent.
7. Handle the received result data carefully and securely, even though the data comes from a partner application.

PartnerUserActivity.java

```
/*
 * Copyright (C) 2012-2026 Japan Smartphone Security Association
 *
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 *
 *      http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */

package org.jssec.android.provider.partneruser;

import org.jssec.android.shared.PkgCertWhitelists;
import org.jssec.android.shared.Utils;

import android.app.Activity;
import android.content.ContentValues;
import android.content.Context;
import android.content.pm.ProviderInfo;
import android.database.Cursor;
import android.net.Uri;
import android.os.Bundle;
import android.view.View;
import android.widget.TextView;

public class PartnerUserActivity extends Activity {

    // Target Content Provider Information
    private static final String AUTHORITY =
            "org.jssec.android.provider.partnerprovider";
    private interface Address {
        public static final String PATH = "addresses";
        public static final Uri CONTENT_URI =
                Uri.parse("content://" + AUTHORITY + "/" + PATH);
    }

    // *** POINT 4 *** Verify if the certificate of the target application has
    // been registered in the own white list.
```

(continues on next page)

(continued from previous page)

```
    private static PkgCertWhitelists sWhitelists = null;
    private static void buildWhitelists(Context context) {
        boolean isdebug = Utils.isDebuggable(context);
        sWhitelists = new PkgCertWhitelists();

        // Register certificate hash value of partner application
        // org.jssec.android.provider.partnerprovider.
        sWhitelists.add("org.jssec.android.provider.partnerprovider", isdebug ?
            // Certificate hash value of "androiddebugkey" in the debug.keystore.
            "0EFB7236 328348A9 89718BAD DF57F544 D5CCB4AE B9DB34BC 1E29DD26␣
↪F77C8255" :
            // Certificate hash value of "partner key" in the keystore.
            "D397D343 A5CBC10F 4EDDEB7C A10062DE 5690984F 1FB9E88B D7B3A7C2␣
↪42E142CA");

        // Register following other partner applications in the same way.
    }

    private static boolean checkPartner(Context context, String pkgname) {
        if (sWhitelists == null) buildWhitelists(context);
        return sWhitelists.test(context, pkgname);
    }

    // Get package name of target content provider.
    private String providerPkgname(Uri uri) {
        String pkgname = null;
        ProviderInfo pi =
            getPackageManager().resolveContentProvider(uri.getAuthority(), 0);
        if (pi != null) pkgname = pi.packageName;
        return pkgname;
    }

    public void onQueryClick(View view) {

        logLine("[Query]");

        // *** POINT 4 *** Verify if the certificate of the target application has
        // been registered in the own white list.
        if (!checkPartner(this, providerPkgname(Address.CONTENT_URI))) {
            logLine("  The target content provider is not served by partner␣
↪applications.");
            return;
        }

        Cursor cursor = null;
        try {
            // *** POINT 5 *** Information that is granted to disclose to partner
            // applications can be sent.
            cursor = getContentResolver().query(Address.CONTENT_URI,
                                                null, null, null, null);

            // *** POINT 6 *** Handle the received result data carefully and
            // securely, even though the data comes from a partner application.
            // Omitted, since this is a sample. Please refer to
            // "3.2 Handling Input Data Carefully and Securely."
            if (cursor == null) {
```

(continues on next page)

(continued from previous page)

```
                logLine("  null cursor");
            } else {
                boolean moved = cursor.moveToFirst();
                while (moved) {
                    logLine(String.format("  %d, %s", cursor.getInt(0),
                                          cursor.getString(1)));
                    moved = cursor.moveToNext();
                }
            }
        }
        finally {
            if (cursor != null) cursor.close();
        }
    }

    public void onInsertClick(View view) {

        logLine("[Insert]");

        // *** POINT 4 *** Verify if the certificate of the target application has
        // been registered in the own white list.
        if (!checkPartner(this, providerPkgname(Address.CONTENT_URI))) {
            logLine("  The target content provider is not served by partner
↪applications.");
            return;
        }

        // *** POINT 5 *** Information that is granted to disclose to partner
        // applications can be sent.
        ContentValues values = new ContentValues();
        values.put("city", "Tokyo");
        Uri uri = getContentResolver().insert(Address.CONTENT_URI, values);

        // *** POINT 6 *** Handle the received result data carefully and securely,
        // even though the data comes from a partner application.
        // Omitted, since this is a sample. Please refer to
        // "3.2 Handling Input Data Carefully and Securely."
        logLine("  uri:" + uri);
    }

    public void onUpdateClick(View view) {

        logLine("[Update]");

        // *** POINT 4 *** Verify if the certificate of the target application has
        // been registered in the own white list.
        if (!checkPartner(this, providerPkgname(Address.CONTENT_URI))) {
            logLine("  The target content provider is not served by partner
↪applications.");
            return;
        }

        // *** POINT 5 *** Information that is granted to disclose to partner
        // applications can be sent.
        ContentValues values = new ContentValues();
        values.put("city", "Tokyo");
```

(continues on next page)

(continued from previous page)

```
        String where = "_id = ?";
        String[] args = { "4" };
        int count =
            getContentResolver().update(Address.CONTENT_URI, values, where, args);

        // *** POINT 6 *** Handle the received result data carefully and securely,
        // even though the data comes from a partner application.
        // Omitted, since this is a sample. Please refer to
        // "3.2 Handling Input Data Carefully and Securely."
        logLine(String.format("  %s records updated", count));
    }

    public void onDeleteClick(View view) {

        logLine("[Delete]");

        // *** POINT 4 *** Verify if the certificate of the target application has
        // been registered in the own white list.
        if (!checkPartner(this, providerPkgname(Address.CONTENT_URI))) {
            logLine("  The target content provider is not served by partner␣
↪applications.");
            return;
        }

        // *** POINT 5 *** Information that is granted to disclose to partner
        // applications can be sent.
        int count = getContentResolver().delete(Address.CONTENT_URI, null, null);

        // *** POINT 6 *** Handle the received result data carefully and securely,
        // even though the data comes from a partner application.
        // Omitted, since this is a sample. Please refer to
        // "3.2 Handling Input Data Carefully and Securely."
        logLine(String.format("  %s records deleted", count));
    }

    private TextView mLogView;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        mLogView = (TextView)findViewById(R.id.logview);
    }

    private void logLine(String line) {
        mLogView.append(line);
        mLogView.append("\n");
    }
}
```

PkgCertWhitelists.java

```
/*
 * Copyright (C) 2012-2026 Japan Smartphone Security Association
 *
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
```

(continues on next page)

(continued from previous page)

```
 * You may obtain a copy of the License at
 *
 *      http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */

package org.jssec.android.shared;

import android.content.pm.PackageManager;
import java.util.HashMap;
import java.util.Map;
import android.content.Context;
import android.os.Build;

import static android.content.pm.PackageManager.CERT_INPUT_SHA256;

public class PkgCertWhitelists {
    private Map<String, String> mWhitelists = new HashMap<String, String>();

    public boolean add(String pkgname, String sha256) {
        if (pkgname == null) return false;
        if (sha256 == null) return false;

        sha256 = sha256.replaceAll(" ", "");
        if (sha256.length() != 64)
            return false;  // SHA-256 -> 32 bytes -> 64 chars
        sha256 = sha256.toUpperCase();
        if (sha256.replaceAll("[0-9A-F]+", "").length() != 0)
            return false;  // found non hex char

        mWhitelists.put(pkgname, sha256);
        return true;
    }

    public boolean test(Context ctx, String pkgname) {
        // Get the correct hash value which corresponds to pkgname.
        String correctHash = mWhitelists.get(pkgname);

        // Compare the actual hash value of pkgname with the correct hash value.
        if (Build.VERSION.SDK_INT >= 28) {
            // ** if API Level >= 28, direct checking is possible
            PackageManager pm = ctx.getPackageManager();
            return pm.hasSigningCertificate(pkgname,
                                            Utils.hex2Bytes(correctHash),
                                            CERT_INPUT_SHA256);
        } else {
            // else use the facility of PkgCert
            return PkgCert.test(ctx, pkgname, correctHash);
        }
    }
}
```

PkgCert.java

```java
/*
 * Copyright (C) 2012-2026 Japan Smartphone Security Association
 *
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 *
 *      http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */

package org.jssec.android.shared;

import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;

import android.content.Context;
import android.content.pm.PackageInfo;
import android.content.pm.PackageManager;
import android.content.pm.PackageManager.NameNotFoundException;
import android.content.pm.Signature;

public class PkgCert {

    public static boolean test(Context ctx, String pkgname, String correctHash) {
        if (correctHash == null) return false;
        correctHash = correctHash.replaceAll(" ", "");
        return correctHash.equals(hash(ctx, pkgname));
    }

    public static String hash(Context ctx, String pkgname) {
        if (pkgname == null) return null;
        try {
            PackageManager pm = ctx.getPackageManager();
            PackageInfo pkginfo =
                pm.getPackageInfo(pkgname, PackageManager.GET_SIGNATURES);
            // Will not handle multiple signatures.
            if (pkginfo.signatures.length != 1) return null;
            Signature sig = pkginfo.signatures[0];
            byte[] cert = sig.toByteArray();
            byte[] sha256 = computeSha256(cert);
            return byte2hex(sha256);
        } catch (NameNotFoundException e) {
            return null;
        }
    }

    private static byte[] computeSha256(byte[] data) {
        try {
            return MessageDigest.getInstance("SHA-256").digest(data);
        } catch (NoSuchAlgorithmException e) {
```

(continues on next page)

(continued from previous page)

```
            return null;
        }
    }

    private static String byte2hex(byte[] data) {
        if (data == null) return null;
        final StringBuilder hexadecimal = new StringBuilder();
        for (final byte b : data) {
            hexadecimal.append(String.format("%02X", b));
        }
        return hexadecimal.toString();
    }
}
```

4.3.1.4 Creating/Using In-house Content Providers

In-house Content Provider is the Content Provider which prohibits to be used by applications other than In house only applications.

Sample code of how to implement an In house only Content Provider is shown below.

Points (Creating a Content Provider):

1. Define an in-house signature permission.
2. Require the in-house signature permission.
3. Explicitly set the exported attribute to true.
4. Verify if the in-house signature permission is defined by an in-house application.
5. Verify the safety of the parameter even if it's a request from In house only application.
6. Sensitive information can be returned since the requesting application is in-house.
7. When exporting an APK, sign the APK with the same developer key as that of the requesting application.

```
AndroidManifest.xml
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
          >

  <!-- *** POINT 1 *** Define an in-house signature permission -->
  <permission
      android:name="org.jssec.android.provider.inhouseprovider.MY_PERMISSION"
      android:protectionLevel="signature" />

  <application
      android:icon="@drawable/ic_launcher"
      android:label="@string/app_name" >

    <!-- *** POINT 2 *** Require the in-house signature permission -->
    <!-- *** POINT 3 *** Explicitly set the exported attribute to true. -->
    <provider
        android:name=".InhouseProvider"
        android:authorities="org.jssec.android.provider.inhouseprovider"
        android:permission="org.jssec.android.provider.inhouseprovider.MY_
↪PERMISSION"
        android:exported="true" />
```

(continues on next page)

(continued from previous page)

```
  </application>
</manifest>
```

InhouseProvider.java

```
/*
 * Copyright (C) 2012-2026 Japan Smartphone Security Association
 *
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 *
 *      http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */

package org.jssec.android.provider.inhouseprovider;

import org.jssec.android.shared.SigPerm;
import org.jssec.android.shared.Utils;

import android.content.ContentProvider;
import android.content.ContentUris;
import android.content.ContentValues;
import android.content.Context;
import android.content.UriMatcher;
import android.database.Cursor;
import android.database.MatrixCursor;
import android.net.Uri;

public class InhouseProvider extends ContentProvider {

    public static final String AUTHORITY =
            "org.jssec.android.provider.inhouseprovider";
    public static final String CONTENT_TYPE =
            "vnd.android.cursor.dir/vnd.org.jssec.contenttype";
    public static final String CONTENT_ITEM_TYPE =
            "vnd.android.cursor.item/vnd.org.jssec.contenttype";

    // Expose the interface that the Content Provider provides.
    public interface Download {
        public static final String PATH = "downloads";
        public static final Uri CONTENT_URI =
                Uri.parse("content://" + AUTHORITY + "/" + PATH);
    }
    public interface Address {
        public static final String PATH = "addresses";
        public static final Uri CONTENT_URI =
                Uri.parse("content://" + AUTHORITY + "/" + PATH);
    }

    // UriMatcher
```

(continues on next page)

(continued from previous page)

```
    private static final int DOWNLOADS_CODE = 1;
    private static final int DOWNLOADS_ID_CODE = 2;
    private static final int ADDRESSES_CODE = 3;
    private static final int ADDRESSES_ID_CODE = 4;
    private static UriMatcher sUriMatcher;
    static {
        sUriMatcher = new UriMatcher(UriMatcher.NO_MATCH);
        sUriMatcher.addURI(AUTHORITY, Download.PATH, DOWNLOADS_CODE);
        sUriMatcher.addURI(AUTHORITY, Download.PATH + "/#", DOWNLOADS_ID_CODE);
        sUriMatcher.addURI(AUTHORITY, Address.PATH, ADDRESSES_CODE);
        sUriMatcher.addURI(AUTHORITY, Address.PATH + "/#", ADDRESSES_ID_CODE);
    }

    // Since this is a sample program, query method returns the following
    // fixed result always without using database.
    private static MatrixCursor sAddressCursor =
            new MatrixCursor(new String[] { "_id", "city" });
    static {
        sAddressCursor.addRow(new String[] { "1", "New York" });
        sAddressCursor.addRow(new String[] { "2", "London" });
        sAddressCursor.addRow(new String[] { "3", "Paris" });
    }
    private static MatrixCursor sDownloadCursor =
            new MatrixCursor(new String[] { "_id", "path" });
    static {
        sDownloadCursor.addRow(new String[] { "1", "/sdcard/downloads/sample.jpg" }
↪);
        sDownloadCursor.addRow(new String[] { "2", "/sdcard/downloads/sample.txt" }
↪);
    }

    // In-house Signature Permission
    private static final String MY_PERMISSION =
            "org.jssec.android.provider.inhouseprovider.MY_PERMISSION";

    // In-house certificate hash value
    private static String sMyCertHash = null;
    private static String myCertHash(Context context) {
        if (sMyCertHash == null) {
            if (Utils.isDebuggable(context)) {
                // Certificate hash value of "androiddebugkey" in the
                // debug.keystore.
                sMyCertHash = "0EFB7236 328348A9 89718BAD DF57F544 D5CCB4AE␣
↪B9DB34BC 1E29DD26 F77C8255";
            } else {
                // Certificate hash value of "my company key" in the keystore.
                sMyCertHash = "D397D343 A5CBC10F 4EDDEB7C A10062DE 5690984F␣
↪1FB9E88B D7B3A7C2 42E142CA";
            }
        }
        return sMyCertHash;
    }

    @Override
    public boolean onCreate() {
        return true;
```

(continues on next page)

(continued from previous page)

```
    }

    @Override
    public String getType(Uri uri) {

        switch (sUriMatcher.match(uri)) {
        case DOWNLOADS_CODE:
        case ADDRESSES_CODE:
            return CONTENT_TYPE;

        case DOWNLOADS_ID_CODE:
        case ADDRESSES_ID_CODE:
            return CONTENT_ITEM_TYPE;

        default:
            throw new IllegalArgumentException("Invalid URI:" + uri);
        }
    }

    @Override
    public Cursor query(Uri uri, String[] projection, String selection,
                        String[] selectionArgs, String sortOrder) {

        // *** POINT 4 *** Verify if the in-house signature permission is defined
        // by an in-house application.
        if (!SigPerm.test(getContext(), MY_PERMISSION, myCertHash(getContext()))) {
            throw new SecurityException("The in-house signature permission is not␣
↪declared by in-house application.");
        }

        // *** POINT 5 *** Handle the received request data carefully and securely,
        // even though the data came from an in-house application.
        // Here, whether uri is within expectations or not, is verified by
        // UriMatcher#match() and switch case.
        // Checking for other parameters are omitted here, due to sample.
        // Refer to "3.2 Handle Input Data Carefully and Securely."

        // *** POINT 6 *** Sensitive information can be returned since the
        // requesting application is in-house.
        // It depends on application whether the query result has sensitive
        // meaning or not.
        switch (sUriMatcher.match(uri)) {
        case DOWNLOADS_CODE:
        case DOWNLOADS_ID_CODE:
            return sDownloadCursor;

        case ADDRESSES_CODE:
        case ADDRESSES_ID_CODE:
            return sAddressCursor;

        default:
            throw new IllegalArgumentException("Invalid URI:" + uri);
        }
    }

    @Override
```

(continues on next page)

(continued from previous page)

```
    public Uri insert(Uri uri, ContentValues values) {

        // *** POINT 4 *** Verify if the in-house signature permission is defined
        // by an in-house application.
        if (!SigPerm.test(getContext(), MY_PERMISSION, myCertHash(getContext()))) {
            throw new SecurityException("The in-house signature permission is not␣
↪declared by in-house application.");
        }

        // *** POINT 5 *** Handle the received request data carefully and securely,
        // even though the data came from an in-house application.
        // Here, whether uri is within expectations or not, is verified by
        // UriMatcher#match() and switch case.
        // Checking for other parameters are omitted here, due to sample.
        // Refer to "3.2 Handle Input Data Carefully and Securely."

        // *** POINT 6 *** Sensitive information can be returned since the
        // requesting application is in-house.
        // It depends on application whether the issued ID has sensitive meaning
        // or not.
        switch (sUriMatcher.match(uri)) {
        case DOWNLOADS_CODE:
            return ContentUris.withAppendedId(Download.CONTENT_URI, 3);

        case ADDRESSES_CODE:
            return ContentUris.withAppendedId(Address.CONTENT_URI, 4);

        default:
            throw new IllegalArgumentException("Invalid URI:" + uri);
        }
    }

    @Override
    public int update(Uri uri, ContentValues values, String selection,
                      String[] selectionArgs) {

        // *** POINT 4 *** Verify if the in-house signature permission is defined
        // by an in-house application.
        if (!SigPerm.test(getContext(), MY_PERMISSION, myCertHash(getContext()))) {
            throw new SecurityException("The in-house signature permission is not␣
↪declared by in-house application.");
        }

        // *** POINT 5 *** Handle the received request data carefully and securely,
        // even though the data came from an in-house application.
        // Here, whether uri is within expectations or not, is verified by
        // UriMatcher#match() and switch case.
        // Checking for other parameters are omitted here, due to sample.
        // Refer to "3.2 Handle Input Data Carefully and Securely."

        // *** POINT 6 *** Sensitive information can be returned since the
        // requesting application is in-house.
        // It depends on application whether the number of updated records has
        // sensitive meaning or not.
        switch (sUriMatcher.match(uri)) {
        case DOWNLOADS_CODE:
```

(continues on next page)

(continued from previous page)

```
            return 5;   // Return number of updated records

        case DOWNLOADS_ID_CODE:
            return 1;

        case ADDRESSES_CODE:
            return 15;

        case ADDRESSES_ID_CODE:
            return 1;

        default:
            throw new IllegalArgumentException("Invalid URI:" + uri);
        }
    }

    @Override
    public int delete(Uri uri, String selection, String[] selectionArgs) {

        // *** POINT 4 *** Verify if the in-house signature permission is defined
        // by an in-house application.
        if (!SigPerm.test(getContext(), MY_PERMISSION, myCertHash(getContext()))) {
            throw new SecurityException("The in-house signature permission is not␣
↪declared by in-house application.");
        }

        // *** POINT 5 *** Handle the received request data carefully and securely,
        // even though the data came from an in-house application.
        // Here, whether uri is within expectations or not, is verified by
        // UriMatcher#match() and switch case.
        // Checking for other parameters are omitted here, due to sample.
        // Refer to "3.2 Handle Input Data Carefully and Securely."

        // *** POINT 6 *** Sensitive information can be returned since the
        // requesting application is in-house.
        // It depends on application whether the number of deleted records has
        // sensitive meaning or not.
        switch (sUriMatcher.match(uri)) {
        case DOWNLOADS_CODE:
            return 10;  // Return number of deleted records

        case DOWNLOADS_ID_CODE:
            return 1;

        case ADDRESSES_CODE:
            return 20;

        case ADDRESSES_ID_CODE:
            return 1;

        default:
            throw new IllegalArgumentException("Invalid URI:" + uri);
        }
    }
}
```

SigPerm.java

```java
/*
 * Copyright (C) 2012-2026 Japan Smartphone Security Association
 *
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 *
 *      http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */

package org.jssec.android.shared;

import android.content.Context;
import android.content.pm.PackageManager;
import android.content.pm.PackageManager.NameNotFoundException;
import android.content.pm.PermissionInfo;
import android.os.Build;

import static android.content.pm.PackageManager.CERT_INPUT_SHA256;

public class SigPerm {

    public static boolean test(Context ctx, String sigPermName,
                               String correctHash) {
        if (correctHash == null) return false;
        correctHash = correctHash.replaceAll(" ", "");
        try {
            // Get the package name of the application which declares a permission
            // named sigPermName.
            PackageManager pm = ctx.getPackageManager();
            PermissionInfo pi =
                pm.getPermissionInfo(sigPermName, PackageManager.GET_META_DATA);
            String pkgname = pi.packageName;
            // Fail if the permission named sigPermName is not a Signature
            // Permission
            if (pi.protectionLevel != PermissionInfo.PROTECTION_SIGNATURE)
                return false;

            // Compare the actual hash value of pkgname with the correct hash
            // value.
            if (Build.VERSION.SDK_INT >= 28) {
                // ** if API Level >= 28, direct check is possible
                return pm.hasSigningCertificate(pkgname,
                                                Utils.hex2Bytes(correctHash),
                                                CERT_INPUT_SHA256);
            } else {
                // else(API Level < 28) use the facility of PkgCert
                return correctHash.equals(PkgCert.hash(ctx, pkgname));
            }
```

(continues on next page)

(continued from previous page)

```
        } catch (NameNotFoundException e) {
            return false;
        }
    }
}
```

PkgCert.java

```
/*
 * Copyright (C) 2012-2026 Japan Smartphone Security Association
 *
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 *
 *      http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */

package org.jssec.android.shared;

import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;

import android.content.Context;
import android.content.pm.PackageInfo;
import android.content.pm.PackageManager;
import android.content.pm.PackageManager.NameNotFoundException;
import android.content.pm.Signature;

public class PkgCert {

    public static boolean test(Context ctx, String pkgname, String correctHash) {
        if (correctHash == null) return false;
        correctHash = correctHash.replaceAll(" ", "");
        return correctHash.equals(hash(ctx, pkgname));
    }

    public static String hash(Context ctx, String pkgname) {
        if (pkgname == null) return null;
        try {
            PackageManager pm = ctx.getPackageManager();
            PackageInfo pkginfo =
                pm.getPackageInfo(pkgname, PackageManager.GET_SIGNATURES);
            // Will not handle multiple signatures.
            if (pkginfo.signatures.length != 1) return null;
            Signature sig = pkginfo.signatures[0];
            byte[] cert = sig.toByteArray();
            byte[] sha256 = computeSha256(cert);
            return byte2hex(sha256);
        } catch (NameNotFoundException e) {
            return null;
```

(continues on next page)

(continued from previous page)

```
        }
    }

    private static byte[] computeSha256(byte[] data) {
        try {
            return MessageDigest.getInstance("SHA-256").digest(data);
        } catch (NoSuchAlgorithmException e) {
            return null;
        }
    }

    private static String byte2hex(byte[] data) {
        if (data == null) return null;
        final StringBuilder hexadecimal = new StringBuilder();
        for (final byte b : data) {
            hexadecimal.append(String.format("%02X", b));
        }
        return hexadecimal.toString();
    }
}
```

*** Point 7 *** When exporting an APK, sign the APK with the same developer key as the requesting application.

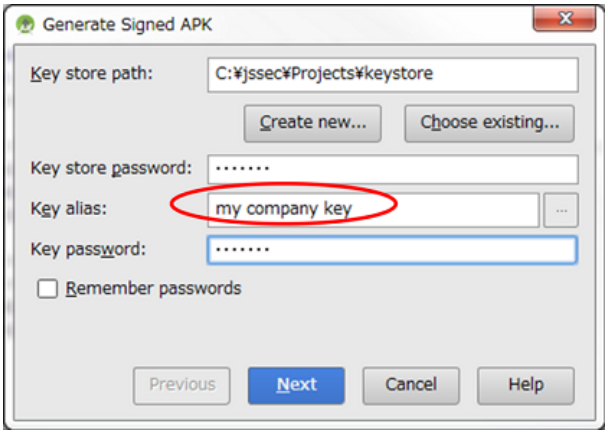


Fig. 4.3.2: Sign the APK with the same developer key as the requesting application

Next is the example of Activity which uses In house only Content Provider.

Point (Using a Content Provider):

8. Declare to use the in-house signature permission.
9. Verify if the in-house signature permission is defined by an in-house application.0
10. Verify if the destination application is signed with the in-house certificate.
11. Sensitive information can be sent since the destination application is in-house one.
12. Handle the received result data carefully and securely, even though the data comes from an in-house application.
13. When exporting an APK, sign the APK with the same developer key as that of the destination application.

AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    >
```

(continues on next page)

(continued from previous page)

```
    <!-- *** POINT 8 *** Declare to use the in-house signature permission. -->
    <uses-permission
        android:name="org.jssec.android.provider.inhouseprovider.MY_PERMISSION" />

    <application
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name" >
        <activity
            android:name=".InhouseUserActivity"
            android:label="@string/app_name"
            android:exported="true" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>
</manifest>
```

InhouseUserActivity.java

```
/*
 * Copyright (C) 2012-2026 Japan Smartphone Security Association
 *
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 *
 *      http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */

package org.jssec.android.provider.inhouseuser;

import org.jssec.android.shared.PkgCert;
import org.jssec.android.shared.SigPerm;
import org.jssec.android.shared.Utils;

import android.app.Activity;
import android.content.ContentValues;
import android.content.Context;
import android.content.pm.PackageManager;
import android.content.pm.ProviderInfo;
import android.database.Cursor;
import android.net.Uri;
import android.os.Bundle;
import android.view.View;
import android.widget.TextView;

public class InhouseUserActivity extends Activity {
```

(continues on next page)

(continued from previous page)

```
    // Target Content Provider Information
    private static final String AUTHORITY =
            "org.jssec.android.provider.inhouseprovider";
    private interface Address {
        public static final String PATH = "addresses";
        public static final Uri CONTENT_URI =
                Uri.parse("content://" + AUTHORITY + "/" + PATH);
    }

    // In-house Signature Permission
    private static final String MY_PERMISSION =
            "org.jssec.android.provider.inhouseprovider.MY_PERMISSION";

    // In-house certificate hash value
    private static String sMyCertHash = null;
    private static String myCertHash(Context context) {
        if (sMyCertHash == null) {
            if (Utils.isDebuggable(context)) {
                // Certificate hash value of "androiddebugkey" in the
                // debug.keystore.
                sMyCertHash = "0EFB7236 328348A9 89718BAD DF57F544 D5CCB4AE␣
↪B9DB34BC 1E29DD26 F77C8255";
            } else {
                // Certificate hash value of "my company key" in the keystore.
                sMyCertHash = "D397D343 A5CBC10F 4EDDEB7C A10062DE 5690984F␣
↪1FB9E88B D7B3A7C2 42E142CA";
            }
        }
        return sMyCertHash;
    }

    // Get package name of target content provider.
    private static String providerPkgname(Context context, Uri uri) {
        String pkgname = null;
        PackageManager pm = context.getPackageManager();
        ProviderInfo pi = pm.resolveContentProvider(uri.getAuthority(), 0);
        if (pi != null) pkgname = pi.packageName;
        return pkgname;
    }

    public void onQueryClick(View view) {

        logLine("[Query]");

        // *** POINT 9 *** Verify if the in-house signature permission is defined
        // by an in-house application.
        if (!SigPerm.test(this, MY_PERMISSION, myCertHash(this))) {
            logLine("  The in-house signature permission is not declared by in-
↪house application.");
            return;
        }

        // *** POINT 10 *** Verify if the destination application is signed with
        // the in-house certificate.
        String pkgname = providerPkgname(this, Address.CONTENT_URI);
        if (!PkgCert.test(this, pkgname, myCertHash(this))) {
```

(continues on next page)

(continued from previous page)

```
            logLine("  The target content provider is not served by in-house␣
↪applications.");
            return;
        }

        Cursor cursor = null;
        try {
            // *** POINT 11 *** Sensitive information can be sent since the
            // destination application is in-house one.
            cursor =
                getContentResolver().query(Address.CONTENT_URI,
                                           null, null, null, null);

            // *** POINT 12 *** Handle the received result data carefully and
            // securely, even though the data comes from an in-house application.
            // Omitted, since this is a sample. Please refer to
            // "3.2 Handling Input Data Carefully and Securely."
            if (cursor == null) {
                logLine("  null cursor");
            } else {
                boolean moved = cursor.moveToFirst();
                while (moved) {
                    logLine(String.format("  %d, %s", cursor.getInt(0),
                                          cursor.getString(1)));
                    moved = cursor.moveToNext();
                }
            }
        }
        finally {
            if (cursor != null) cursor.close();
        }
    }

    public void onInsertClick(View view) {

        logLine("[Insert]");

        // *** POINT 9 *** Verify if the in-house signature permission is defined
        // by an in-house application.
        String correctHash = myCertHash(this);
        if (!SigPerm.test(this, MY_PERMISSION, correctHash)) {
            logLine("  The in-house signature permission is not declared by in-
↪house application.");
            return;
        }

        // *** POINT 10 *** Verify if the destination application is signed with
        // the in-house certificate.
        String pkgname = providerPkgname(this, Address.CONTENT_URI);
        if (!PkgCert.test(this, pkgname, correctHash)) {
            logLine("  The target content provider is not served by in-house␣
↪applications.");
            return;
        }

        // *** POINT 11 *** Sensitive information can be sent since the
```

(continues on next page)

(continued from previous page)

```
        // destination application is in-house one.
        ContentValues values = new ContentValues();
        values.put("city", "Tokyo");
        Uri uri = getContentResolver().insert(Address.CONTENT_URI, values);

        // *** POINT 12 *** Handle the received result data carefully and securely,
        // even though the data comes from an in-house application.
        // Omitted, since this is a sample. Please refer to
        // "3.2 Handling Input Data Carefully and Securely."
        logLine("  uri:" + uri);
    }

    public void onUpdateClick(View view) {

        logLine("[Update]");

        // *** POINT 9 *** Verify if the in-house signature permission is defined
        // by an in-house application.
        String correctHash = myCertHash(this);
        if (!SigPerm.test(this, MY_PERMISSION, correctHash)) {
            logLine("  The in-house signature permission is not declared by in-
↪house application.");
            return;
        }

        // *** POINT 10 *** Verify if the destination application is signed with
        // the in-house certificate.
        String pkgname = providerPkgname(this, Address.CONTENT_URI);
        if (!PkgCert.test(this, pkgname, correctHash)) {
            logLine("  The target content provider is not served by in-house␣
↪applications.");
            return;
        }

        // *** POINT 11 *** Sensitive information can be sent since the
        // destination application is in-house one.
        ContentValues values = new ContentValues();
        values.put("city", "Tokyo");
        String where = "_id = ?";
        String[] args = { "4" };
        int count =
            getContentResolver().update(Address.CONTENT_URI, values, where, args);

        // *** POINT 12 *** Handle the received result data carefully and securely,
        // even though the data comes from an in-house application.
        // Omitted, since this is a sample. Please refer to
        // "3.2 Handling Input Data Carefully and Securely."
        logLine(String.format("  %s records updated", count));
    }

    public void onDeleteClick(View view) {

        logLine("[Delete]");

        // *** POINT 9 *** Verify if the in-house signature permission is defined
        // by an in-house application.
```

(continues on next page)

(continued from previous page)

```
        String correctHash = myCertHash(this);
        if (!SigPerm.test(this, MY_PERMISSION, correctHash)) {
            logLine("  The target content provider is not served by in-house␣
↪applications.");
            return;
        }

        // *** POINT 10 *** Verify if the destination application is signed with
        // the in-house certificate.
        String pkgname = providerPkgname(this, Address.CONTENT_URI);
        if (!PkgCert.test(this, pkgname, correctHash)) {
            logLine("  The target content provider is not served by in-house␣
↪applications.");
            return;
        }

        // *** POINT 11 *** Sensitive information can be sent since the
        // destination application is in-house one.
        int count = getContentResolver().delete(Address.CONTENT_URI, null, null);

        // *** POINT 12 *** Handle the received result data carefully and securely,
        // even though the data comes from an in-house application.
        // Omitted, since this is a sample. Please refer to
        // "3.2 Handling Input Data Carefully and Securely."
        logLine(String.format("  %s records deleted", count));
    }

    private TextView mLogView;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        mLogView = (TextView)findViewById(R.id.logview);
    }

    private void logLine(String line) {
        mLogView.append(line);
        mLogView.append("\n");
    }
}
```

SigPerm.java

```
/*
 * Copyright (C) 2012-2026 Japan Smartphone Security Association
 *
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 *
 *      http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
```

(continues on next page)

(continued from previous page)

```
 * limitations under the License.
 */

package org.jssec.android.shared;

import android.content.Context;
import android.content.pm.PackageManager;
import android.content.pm.PackageManager.NameNotFoundException;
import android.content.pm.PermissionInfo;
import android.os.Build;

import static android.content.pm.PackageManager.CERT_INPUT_SHA256;

public class SigPerm {

    public static boolean test(Context ctx, String sigPermName,
                               String correctHash) {
        if (correctHash == null) return false;
        correctHash = correctHash.replaceAll(" ", "");
        try {
            // Get the package name of the application which declares a permission
            // named sigPermName.
            PackageManager pm = ctx.getPackageManager();
            PermissionInfo pi =
                pm.getPermissionInfo(sigPermName, PackageManager.GET_META_DATA);
            String pkgname = pi.packageName;
            // Fail if the permission named sigPermName is not a Signature
            // Permission
            if (pi.protectionLevel != PermissionInfo.PROTECTION_SIGNATURE)
                return false;

            // Compare the actual hash value of pkgname with the correct hash
            // value.
            if (Build.VERSION.SDK_INT >= 28) {
                // ** if API Level >= 28, direct check is possible
                return pm.hasSigningCertificate(pkgname,
                                                Utils.hex2Bytes(correctHash),
                                                CERT_INPUT_SHA256);
            } else {
                // else(API Level < 28) use the facility of PkgCert
                return correctHash.equals(PkgCert.hash(ctx, pkgname));
            }

        } catch (NameNotFoundException e) {
            return false;
        }
    }
}
```

PkgCert.java

```
/*
 * Copyright (C) 2012-2026 Japan Smartphone Security Association
 *
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
```

(continues on next page)

(continued from previous page)

```
 *
 *      http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */

package org.jssec.android.shared;

import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;

import android.content.Context;
import android.content.pm.PackageInfo;
import android.content.pm.PackageManager;
import android.content.pm.PackageManager.NameNotFoundException;
import android.content.pm.Signature;

public class PkgCert {

    public static boolean test(Context ctx, String pkgname, String correctHash) {
        if (correctHash == null) return false;
        correctHash = correctHash.replaceAll(" ", "");
        return correctHash.equals(hash(ctx, pkgname));
    }

    public static String hash(Context ctx, String pkgname) {
        if (pkgname == null) return null;
        try {
            PackageManager pm = ctx.getPackageManager();
            PackageInfo pkginfo =
                pm.getPackageInfo(pkgname, PackageManager.GET_SIGNATURES);
            // Will not handle multiple signatures.
            if (pkginfo.signatures.length != 1) return null;
            Signature sig = pkginfo.signatures[0];
            byte[] cert = sig.toByteArray();
            byte[] sha256 = computeSha256(cert);
            return byte2hex(sha256);
        } catch (NameNotFoundException e) {
            return null;
        }
    }

    private static byte[] computeSha256(byte[] data) {
        try {
            return MessageDigest.getInstance("SHA-256").digest(data);
        } catch (NoSuchAlgorithmException e) {
            return null;
        }
    }

    private static String byte2hex(byte[] data) {
        if (data == null) return null;
```

(continues on next page)

(continued from previous page)

```
        final StringBuilder hexadecimal = new StringBuilder();
        for (final byte b : data) {
            hexadecimal.append(String.format("%02X", b));
        }
        return hexadecimal.toString();
    }
}
```

*** Point 13 *** When exporting an APK, sign the APK with the same developer key as that of the destination application.

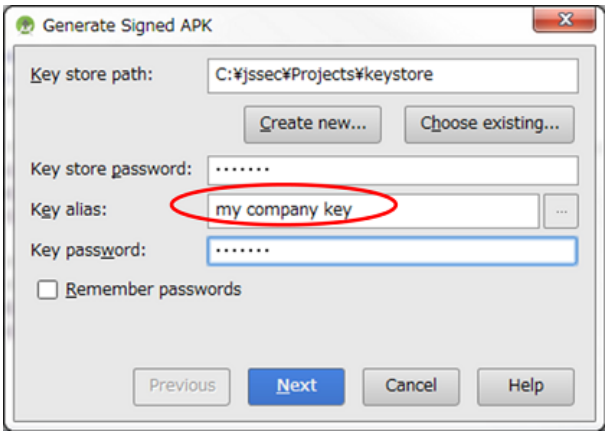


Fig. 4.3.3: Sign the APK with the same developer key as the destination application

4.3.1.5 Creating/Using Temporary permit Content Providers

Temporary permit Content Provider is basically a private Content Provider, but this permits the particular applications to access the particular URI. By sending an Intent which special flag is specified to the target applications, temporary access permission is provided to those applications. Contents provider side application can give the access permission actively to other applications, and it can also give access permission passively to the application which claims the temporary access permission.

Sample code of how to implement a temporary permit Content Provider is shown below.

Points (Creating a Content Provider):

1. Explicitly set the exported attribute to false.
2. Specify the path to grant access temporarily with the grant-uri-permission.
3. Handle the received request data carefully and securely, even though the data comes from the application granted access temporarily.
4. Information that is granted to disclose to the temporary access applications can be returned.
5. Specify URI for the intent to grant temporary access.
6. Specify access rights for the intent to grant temporary access.
7. Send the explicit intent to an application to grant temporary access.
8. Return the intent to the application that requests temporary access.

```
AndroidManifest.xml
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
          >
```

(continues on next page)

(continued from previous page)

```
  <application
      android:icon="@drawable/ic_launcher"
      android:label="@string/app_name" >

    <activity
        android:name=".TemporaryActiveGrantActivity"
        android:label="@string/app_name"
        android:exported="true" >
      <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER" />
      </intent-filter>
    </activity>

    <!-- Temporary Content Provider -->
    <!-- *** POINT 1 *** Explicitly set the exported attribute to false. -->
    <provider
        android:name=".TemporaryProvider"
        android:authorities="org.jssec.android.provider.temporaryprovider"
        android:exported="false" >

      <!-- *** POINT 2 *** Specify the path to grant access temporarily with the␣
↪grant-uri-permission. -->
      <grant-uri-permission android:path="/addresses" />

    </provider>

    <activity
        android:name=".TemporaryPassiveGrantActivity"
        android:label="@string/app_name"
        android:exported="true" />
  </application>
</manifest>
```

TemporaryProvider.java

```
/*
 * Copyright (C) 2012-2026 Japan Smartphone Security Association
 *
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 *
 *      http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */

package org.jssec.android.provider.temporaryprovider;

import android.content.ContentProvider;
import android.content.ContentUris;
```

(continues on next page)

(continued from previous page)

```
import android.content.ContentValues;
import android.content.UriMatcher;
import android.database.Cursor;
import android.database.MatrixCursor;
import android.net.Uri;

public class TemporaryProvider extends ContentProvider {
    public static final String AUTHORITIY =
            "org.jssec.android.provider.temporaryprovider";
    public static final String CONTENT_TYPE =
            "vnd.android.cursor.dir/vnd.org.jssec.contenttype";
    public static final String CONTENT_ITEM_TYPE =
            "vnd.android.cursor.item/vnd.org.jssec.contenttype";

    // Expose the interface that the Content Provider provides.
    public interface Download {
        public static final String PATH = "downloads";
        public static final Uri CONTENT_URI =
                Uri.parse("content://" + AUTHORITIY + "/" + PATH);
    }
    public interface Address {
        public static final String PATH = "addresses";
        public static final Uri CONTENT_URI =
                Uri.parse("content://" + AUTHORITIY + "/" + PATH);
    }

    // UriMatcher
    private static final int DOWNLOADS_CODE = 1;
    private static final int DOWNLOADS_ID_CODE = 2;
    private static final int ADDRESSES_CODE = 3;
    private static final int ADDRESSES_ID_CODE = 4;
    private static UriMatcher sUriMatcher;
    static {
        sUriMatcher = new UriMatcher(UriMatcher.NO_MATCH);
        sUriMatcher.addURI(AUTHORITIY, Download.PATH, DOWNLOADS_CODE);
        sUriMatcher.addURI(AUTHORITIY, Download.PATH + "/#", DOWNLOADS_ID_CODE);
        sUriMatcher.addURI(AUTHORITIY, Address.PATH, ADDRESSES_CODE);
        sUriMatcher.addURI(AUTHORITIY, Address.PATH + "/#", ADDRESSES_ID_CODE);
    }

    // Since this is a sample program,
    // query method returns the following fixed result always without using
    // database.
    private static MatrixCursor sAddressCursor =
            new MatrixCursor(new String[] { "_id", "city" });
    static {
        sAddressCursor.addRow(new String[] { "1", "New York" });
        sAddressCursor.addRow(new String[] { "2", "London" });
        sAddressCursor.addRow(new String[] { "3", "Paris" });
    }
    private static MatrixCursor sDownloadCursor =
            new MatrixCursor(new String[] { "_id", "path" });
    static {
        sDownloadCursor.addRow(new String[] { "1", "/sdcard/downloads/sample.jpg" }
↪);
        sDownloadCursor.addRow(new String[] { "2", "/sdcard/downloads/sample.txt" }
```

(continues on next page)

(continued from previous page)

```
↪);
    }

    @Override
    public boolean onCreate() {
        return true;
    }

    @Override
    public String getType(Uri uri) {

        switch (sUriMatcher.match(uri)) {
        case DOWNLOADS_CODE:
        case ADDRESSES_CODE:
            return CONTENT_TYPE;

        case DOWNLOADS_ID_CODE:
        case ADDRESSES_ID_CODE:
            return CONTENT_ITEM_TYPE;

        default:
            throw new IllegalArgumentException("Invalid URI:" + uri);
        }
    }

    @Override
    public Cursor query(Uri uri, String[] projection, String selection,
                        String[] selectionArgs, String sortOrder) {

        // *** POINT 3 *** Handle the received request data carefully and securely,
        // even though the data comes from the application granted access
        // temporarily.
        // Here, whether uri is within expectations or not, is verified by
        // UriMatcher#match() and switch case.
        // Checking for other parameters are omitted here, due to sample.
        // Please refer to "3.2 Handle Input Data Carefully and Securely."

        // *** POINT 4 *** Information that is granted to disclose to the
        // temporary access applications can be returned.
        // It depends on application whether the query result can be disclosed
        // or not.
        switch (sUriMatcher.match(uri)) {
        case DOWNLOADS_CODE:
        case DOWNLOADS_ID_CODE:
            return sDownloadCursor;

        case ADDRESSES_CODE:
        case ADDRESSES_ID_CODE:
            return sAddressCursor;

        default:
            throw new IllegalArgumentException("Invalid URI:" + uri);
        }
    }

    @Override
```

(continues on next page)

(continued from previous page)

```
    public Uri insert(Uri uri, ContentValues values) {

        // *** POINT 3 *** Handle the received request data carefully and securely,
        // even though the data comes from the application granted access
        // temporarily.
        // Here, whether uri is within expectations or not, is verified by
        // UriMatcher#match() and switch case.
        // Checking for other parameters are omitted here, due to sample.
        // Please refer to "3.2 Handle Input Data Carefully and Securely."

        // *** POINT 4 *** Information that is granted to disclose to the
        // temporary access applications can be returned.
        // It depends on application whether the issued ID has sensitive meaning
        // or not.
        switch (sUriMatcher.match(uri)) {
        case DOWNLOADS_CODE:
            return ContentUris.withAppendedId(Download.CONTENT_URI, 3);

        case ADDRESSES_CODE:
            return ContentUris.withAppendedId(Address.CONTENT_URI, 4);

        default:
            throw new IllegalArgumentException("Invalid URI:" + uri);
        }
    }

    @Override
    public int update(Uri uri, ContentValues values, String selection,
                      String[] selectionArgs) {

        // *** POINT 3 *** Handle the received request data carefully and securely,
        // even though the data comes from the application granted access
        // temporarily.
        // Here, whether uri is within expectations or not, is verified by
        // UriMatcher#match() and switch case.
        // Checking for other parameters are omitted here, due to sample.
        // Please refer to "3.2 Handle Input Data Carefully and Securely."

        // *** POINT 4 *** Information that is granted to disclose to the
        // temporary access applications can be returned.
        // It depends on application whether the number of updated records has
        // sensitive meaning or not.
        switch (sUriMatcher.match(uri)) {
        case DOWNLOADS_CODE:
            return 5;   // Return number of updated records

        case DOWNLOADS_ID_CODE:
            return 1;

        case ADDRESSES_CODE:
            return 15;

        case ADDRESSES_ID_CODE:
            return 1;

        default:
```

(continues on next page)

(continued from previous page)

```
            throw new IllegalArgumentException("Invalid URI:" + uri);
        }
    }

    @Override
    public int delete(Uri uri, String selection, String[] selectionArgs) {

        // *** POINT 3 *** Handle the received request data carefully and securely,
        // even though the data comes from the application granted access
        // temporarily.
        // Here, whether uri is within expectations or not, is verified by
        // UriMatcher#match() and switch case.
        // Checking for other parameters are omitted here, due to sample.
        // Please refer to "3.2 Handle Input Data Carefully and Securely."

        // *** POINT 4 *** Information that is granted to disclose to the
        // temporary access applications can be returned.
        // It depends on application whether the number of deleted records has
        // sensitive meaning or not.
        switch (sUriMatcher.match(uri)) {
        case DOWNLOADS_CODE:
            return 10;  // Return number of deleted records

        case DOWNLOADS_ID_CODE:
            return 1;

        case ADDRESSES_CODE:
            return 20;

        case ADDRESSES_ID_CODE:
            return 1;

        default:
            throw new IllegalArgumentException("Invalid URI:" + uri);
        }
    }
}
```

TemporaryActiveGrantActivity.java

```
/*
 * Copyright (C) 2012-2026 Japan Smartphone Security Association
 *
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 *
 *      http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */

package org.jssec.android.provider.temporaryprovider;
```

(continues on next page)

(continued from previous page)

```
import android.app.Activity;
import android.content.ActivityNotFoundException;
import android.content.Intent;
import android.os.Bundle;
import android.view.View;
import android.widget.Toast;

public class TemporaryActiveGrantActivity extends Activity {

    // User Activity Information
    private static final String TARGET_PACKAGE =
            "org.jssec.android.provider.temporaryuser";
    private static final String TARGET_ACTIVITY =
            "org.jssec.android.provider.temporaryuser.TemporaryUserActivity";

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.active_grant);
    }

    // In the case that Content Provider application grants access permission to
    // other application actively.
    public void onSendClick(View view) {
        try {
            Intent intent = new Intent();

            // *** POINT 5 *** Specify URI for the intent to grant temporary
            // access.
            intent.setData(TemporaryProvider.Address.CONTENT_URI);

            // *** POINT 6 *** Specify access rights for the intent to grant
            // temporary access.
            intent.setFlags(Intent.FLAG_GRANT_READ_URI_PERMISSION);

            // *** POINT 7 *** Send the explicit intent to an application to grant
            // temporary access.
            intent.setClassName(TARGET_PACKAGE, TARGET_ACTIVITY);
            startActivity(intent);

        } catch (ActivityNotFoundException e) {
            Toast.makeText(this,
                           "User Activity not found.", Toast.LENGTH_LONG).show();
        }
    }
}
```

TemporaryPassiveGrantActivity.java

```
/*
 * Copyright (C) 2012-2026 Japan Smartphone Security Association
 *
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 *
```

(continues on next page)

(continued from previous page)

```
 *      http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */

package org.jssec.android.provider.temporaryprovider;

import android.app.Activity;
import android.content.Intent;
import android.os.Bundle;
import android.view.View;

public class TemporaryPassiveGrantActivity extends Activity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.passive_grant);
    }

    // In the case that Content Provider application passively grants access
    // permission to the application that requested Content Provider access.
    public void onGrantClick(View view) {
        Intent intent = new Intent();

        // *** POINT 5 *** Specify URI for the intent to grant temporary access.
        intent.setData(TemporaryProvider.Address.CONTENT_URI);

        // *** POINT 6 *** Specify access rights for the intent to grant temporary
        // access.
        intent.setFlags(Intent.FLAG_GRANT_READ_URI_PERMISSION);

        // *** POINT 8 *** Return the intent to the application that requests
        // temporary access.
        setResult(Activity.RESULT_OK, intent);
        finish();
    }

    public void onCloseClick(View view) {
        finish();
    }
}
```

Next is the example of temporary permit Content Provider.

Points (Using a Content Provider):

9. Do not send sensitive information.
10. When receiving a result, handle the result data carefully and securely.

TemporaryUserActivity.java

```
/*
 * Copyright (C) 2012-2026 Japan Smartphone Security Association
 *
```

(continues on next page)

(continued from previous page)

```
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 *
 *      http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */

package org.jssec.android.provider.temporaryuser;

import android.app.Activity;
import android.content.ActivityNotFoundException;
import android.content.Intent;
import android.content.pm.ProviderInfo;
import android.database.Cursor;
import android.net.Uri;
import android.os.Bundle;
import android.view.View;
import android.widget.TextView;

public class TemporaryUserActivity extends Activity {

    // Information of the Content Provider's Activity to request temporary content
    // provider access.
    private static final String TARGET_PACKAGE =
        "org.jssec.android.provider.temporaryprovider";
    private static final String TARGET_ACTIVITY =
        "org.jssec.android.provider.temporaryprovider.TemporaryPassiveGrantActivity
↪";

    // Target Content Provider Information
    private static final String AUTHORITY =
        "org.jssec.android.provider.temporaryprovider";
    private interface Address {
        public static final String PATH = "addresses";
        public static final Uri CONTENT_URI =
                Uri.parse("content://" + AUTHORITY + "/" + PATH);
    }

    private static final int REQUEST_CODE = 1;

    public void onQueryClick(View view) {

        logLine("[Query]");

        Cursor cursor = null;
        try {
            if (!providerExists(Address.CONTENT_URI)) {
                logLine("  Content Provider doesn't exist.");
                return;
            }
```

(continues on next page)

(continued from previous page)

```
            // *** POINT 9 *** Do not send sensitive information.
            // If no problem when the information is taken by malware, it can be
            // included in the request.
            cursor = getContentResolver().query(Address.CONTENT_URI,
                                                null, null, null, null);

            // *** POINT 10 *** When receiving a result, handle the result data
            // carefully and securely.
            // Omitted, since this is a sample. Please refer to
            // "3.2 Handling Input Data Carefully and Securely."
            if (cursor == null) {
                logLine("  null cursor");
            } else {
                boolean moved = cursor.moveToFirst();
                while (moved) {
                    logLine(String.format("  %d, %s", cursor.getInt(0),
                                          cursor.getString(1)));
                    moved = cursor.moveToNext();
                }
            }
        } catch (SecurityException ex) {
            logLine("  Exception:" + ex.getMessage());
        }
        finally {
            if (cursor != null) cursor.close();
        }
    }

    // In the case that this application requests temporary access to the Content
    // Provider and the Content Provider passively grants temporary access
    // permission to this application.
    public void onGrantRequestClick(View view) {
        Intent intent = new Intent();
        intent.setClassName(TARGET_PACKAGE, TARGET_ACTIVITY);
        try {
            startActivityForResult(intent, REQUEST_CODE);
        } catch (ActivityNotFoundException e) {
            logLine("Content Provider's Activity not found.");
        }
    }

    private boolean providerExists(Uri uri) {
        ProviderInfo pi =
            getPackageManager().resolveContentProvider(uri.getAuthority(), 0);
        return (pi != null);
    }

    private TextView mLogView;

    // In the case that the Content Provider application grants temporary access
    // to this application actively.
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
```

(continues on next page)

(continued from previous page)

```
        mLogView = (TextView)findViewById(R.id.logview);
    }

    private void logLine(String line) {
        mLogView.append(line);
        mLogView.append("\n");
    }
}
```

4.3.2 Rule Book

Be sure to follow the rules below when Implementing or using a content provider.

1. *Content Provider that Is Used Only in an Application Must Be Set as Private (Required)*
2. *Handle the Received Request Parameter Carefully and Securely (Required)*
3. *Use an In-house Defined Signature Permission after Verifying that it is Defined by an In-house Application (Required)*
4. *When Returning a Result, Pay Attention to the Possibility of Information Leakage of that Result from the Destination Application (Required)*
5. *When Providing an Asset Secondarily, the Asset should be Protected with the Same Level of Protection (Required)*

And user side should follow the below rules, too.

6. *Handle the Returned Result Data from the Content Provider Carefully and Securely (Required)*

4.3.2.1 Content Provider that Is Used Only in an Application Must Be Set as Private (Required)

Content Provider which is used only in a single application is not necessary to be accessed by other applications, and the access which attacks the Content Provider is not often considered by developers. A Content Provider is basically the system to share data, so it's handled as public by default. A Content Provider which is used only in a single application should be set as private explicitly, and it should be a private Content Provider. In Android 2.3.1 (API Level 9) or later, a Content Provider can be set as private by specifying android:exported="false" in provider element.

```
AndroidManifest.xml
    <!-- 4.3.1.1 - *** POINT 1 *** Explicitly set the exported attribute to false.␣
↪-->
    <provider
        android:name=".PrivateProvider"
        android:authorities="org.jssec.android.provider.privateprovider"
        android:exported="false" />
```

4.3.2.2 Handle the Received Request Parameter Carefully and Securely (Required)

Risks differ depending on the types of Content Providers, but when processing request parameters, the first thing you should do is input validation.

Although each method of a Content Provider has the interface which is supposed to receive the component parameter of SQL statement, actually it simply hands over the arbitrary character string in the system, so it's necessary to pay attention that Contents Provider side needs to suppose the case that unexpected parameter may be provided.

Since Public Content Providers can receive requests from untrusted sources, they can be attacked by malware. On the other hand, Private Content Providers will never receive any requests from other applications directly, but it is possible that a Public Activity in the targeted application may forward a malicious Intent to a Private Content Provider so you should not assume that Private Content Providers cannot receive any malicious input.

Since other Content Providers also have the risk of a malicious intent being forwarded to them as well, it is necessary to perform input validation on these requests as well.

Please refer to "3.2. *Handling Input Data Carefully and Securely*".

4.3.2.3 Use an In-house Defined Signature Permission after Verifying that it is Defined by an In-house Application (Required)

Make sure to protect your in-house Content Providers by defining an in-house signature permission when creating the Content Provider. Since defining a permission in the AndroidManifest.xml file or declaring a permission request does not provide adequate security, please be sure to refer to "5.2.1.2. *How to Communicate Between In-house Applications with In-house-defined Signature Permission*."

4.3.2.4 When Returning a Result, Pay Attention to the Possibility of Information Leakage of that Result from the Destination Application (Required)

In case of query() or insert(), Cursor or Uri is returned to the request sending application as a result information. When sensitive information is included in the result information, the information may be leaked from the destination application. In case of update() or delete(), number of updated/deleted records is returned to the request sending application as a result information. In rare cases, depending on some application specs, the number of updated/deleted records has the sensitive meaning, so please pay attention to this.

4.3.2.5 When Providing an Asset Secondarily, the Asset should be Protected with the Same Level of Protection (Required)

When an information or function asset, which is protected by a permission, is provided to another application secondhand, you need to make sure that it has the same required permissions needed to access the asset. In the Android OS permission security model, only an application that has been granted proper permissions can directly access a protected asset. However, there is a loophole because an application with permissions to an asset can act as a proxy and allow access to an unprivileged application. Substantially this is the same as re-delegating a permission, so it is referred to as the "Permission Re-delegation" problem. Please refer to "5.2.3.4. *Permission Re-delegation Problem*."

4.3.2.6 Handle the Returned Result Data from the Content Provider Carefully and Securely (Required)

Risks differ depending on the types of Content Provider, but when processing a result data, the first thing you should do is input validation.

In case that the destination Content Provider is a public Content Provider, Malware which masquerades as the public Content Provider may return the attack result data. On the other hand, in case that the destination Content Provider is a private Content Provider, it is less risk because it receives the result data from the same application, but you should not assume that private Content Providers cannot receive any malicious input. Since other Content Providers also have the risk of a malicious data being returned to them as well, it is necessary to perform input validation on that result data as well.

Please refer to "3.2. *Handling Input Data Carefully and Securely*"

4.3.3 Advanced

4.3.3.1 Content URI Permission Management

Prior to Android 15, there were certain limitations in permission management for content URIs. Permission checks when sharing content between apps relied primarily on flags such as Intent.FLAG_GRANT_READ_URI_PERMISSION, lacking system-level enforcement and detailed checking mechanisms. Additionally, identifying the calling app was difficult, making accurate verification of permissions challenging.

To resolve these issues, a new content URI permission check API was introduced in Android 15, significantly strengthening security in data sharing between applications.

The following shows the implementation method for content URI permission management utilizing the new features introduced in Android 15.

Content-Providing App

The content-providing app performs permission checks for content URIs and permits access only from apps that possess the appropriate permissions.

```
class MainActivity : ComponentActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        enableEdgeToEdge()
        setContent {
            MyApplicationTheme {
                Scaffold(modifier = Modifier.fillMaxSize()) { innerPadding ->
                    ContentUriPermissionDemo(
                        modifier = Modifier.padding(innerPadding)
                    )
                }
            }
        }
    }
}

@Composable
fun ContentUriPermissionDemo(modifier: Modifier = Modifier) {
    val context = LocalContext.current
    var selectedUri by remember { mutableStateOf<Uri?>(null) }
    var permissionInfo by remember { mutableStateOf("Permission information␣
↪displayed here") }

    val pickImageLauncher = rememberLauncherForActivityResult(
        ActivityResultContracts.GetContent()
    ) { uri: Uri? ->
        uri?.let {
            selectedUri = it
            if (Build.VERSION.SDK_INT >= 35) {
                // Check permissions using the new API in Android 15
                val flags = context.checkContentUriPermissionFull(
                    it,
                    Process.myPid(),
                    Process.myUid(),
                    Intent.FLAG_GRANT_READ_URI_PERMISSION or Intent.FLAG_GRANT_
↪WRITE_URI_PERMISSION
                )
                val readPermission = flags and Intent.FLAG_GRANT_READ_URI_
↪PERMISSION != 0
                val writePermission = flags and Intent.FLAG_GRANT_WRITE_URI_
↪PERMISSION != 0
                permissionInfo = "URI: $it\n" +
                        "Read permission: $readPermission\n" +
                        "Write permission: $writePermission\n" +
                        "Permission flags: $flags"
            } else {
                permissionInfo = "Android 15 or higher required"
            }
```

(continues on next page)

(continued from previous page)

```
        }
    }

    Column(
        modifier = modifier.fillMaxSize().padding(16.dp),
        horizontalAlignment = Alignment.CenterHorizontally,
        verticalArrangement = Arrangement.spacedBy(16.dp)
    ) {
        Button(onClick = { pickImageLauncher.launch("image/*") }) {
            Text("Select Image")
        }

        Text(
            text = permissionInfo,
            style = MaterialTheme.typography.bodyMedium
        )
    }
}
```

```
<activity
    android:name=".ReceiverActivity"
    android:exported="true"
    android:requireContentUriPermissionFromCaller="read">
    <intent-filter>
        <action android:name="android.intent.action.VIEW" />
        <category android:name="android.intent.category.DEFAULT" />
        <data android:scheme="content" android:mimeType="image/*" />
        <data android:scheme="file" android:mimeType="image/*" />
    </intent-filter>
</activity>
```

Content-Using App

The content-using app requests appropriate permissions when accessing content URIs of other apps, and shares content URIs in accordance with the requirements of the destination app.

```
class ReceiverActivity : ComponentActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        enableEdgeToEdge()
        setContent {
            MyApplicationTheme {
                Scaffold(modifier = Modifier.fillMaxSize()) { innerPadding ->
                    ReceiverContent(
                        modifier = Modifier.padding(innerPadding)
                    )
                }
            }
        }
    }
}

@Composable
fun ReceiverContent(modifier: Modifier = Modifier) {
    val context = LocalContext.current as ComponentActivity
    val contentUri = context.intent.data
```

(continues on next page)

(continued from previous page)

```
    // Obtain ComponentCaller outside Composable
    val callerInfo = remember {
        if (Build.VERSION.SDK_INT >= 35) {
            try {
                context.initialCaller
            } catch (e: Exception) {
                null
            }
        } else null
    }

    Column(
        modifier = modifier
            .fillMaxSize()
            .padding(16.dp),
        horizontalAlignment = Alignment.CenterHorizontally,
        verticalArrangement = Arrangement.spacedBy(16.dp)
    ) {
        Text(
            text = "Received Content URI",
            style = MaterialTheme.typography.headlineSmall
        )

        if (contentUri != null) {
            Text("Content URI: $contentUri")

            if (Build.VERSION.SDK_INT >= 35) {
                if (callerInfo != null) {
                    Text(
                        text = "Caller Information:\n" +
                                "Package Name: ${callerInfo.getPackage()}\n" +
                                "User ID: ${callerInfo.getUid()}",
                        style = MaterialTheme.typography.bodyMedium
                    )
                } else {
                    Text(
                        text = "Failed to obtain caller information",
                        style = MaterialTheme.typography.bodyMedium
                    )
                }
            } else {
                Text(
                    text = "ComponentCaller requires Android 15 or higher",
                    style = MaterialTheme.typography.bodyMedium
                )
            }
        } else {
            Text(
                text = "No content URI",
                style = MaterialTheme.typography.bodyMedium
            )
        }
    }
}
```

Through detailed permission checks using the Context.checkContentUriPermissionFull API, system-level permission enforcement via the requireContentUriPermissionFromCaller attribute, and the calling app identification feature

provided by the ComponentCaller class, safer and more transparent content sharing has become possible.

By properly utilizing these features, it is possible to reduce the risk of information leakage from content providers and protect user privacy while achieving efficient data sharing between apps.

However, the requireContentUriPermissionFromCaller attribute is a non-public API designated as @hide in Android's internal implementation. Therefore, there is a possibility of future specification changes or deprecation. Although it can currently be used by specifying the value in the manifest, there is no guarantee that it can be used stably in the future.

4.3.3.2 Tightening of Contacts Provider (CP2)

For applications targeting Android 17 or higher, access restrictions and specification changes are enforced on Contacts Provider (CP2).

First, when accessing the ContactsContract.Data table using temporary URI permissions (such as via Contact Picker) without holding the READ_CONTACTS permission, strict SQL query verification is applied. Queries determined to be incompatible during this verification are rejected, and exceptions such as IllegalArgumentException are thrown.

Second, for the purpose of enhancing privacy, sensitive data fields containing Personally Identifiable Information (PII) (such as ACCOUNT_NAME, ACCOUNT_TYPE, and ACCOUNT_TYPE_AND_DATA_SET) are restricted or removed from data views. Directly specifying these columns in a query results in an exception being thrown or data being unretrievable. To handle this restriction, it is necessary to revise the design to retrieve only RAW_CONTACT_ID from the Data table and use it as a key to query the ContactsContract.RawContacts table separately.

The following shows non-recommended implementations and revised implementation examples for the Contacts Provider restriction enhancements in Android 17.

Traditional Non-Recommended Implementation (In Android 17, an exception occurs or data cannot be retrieved)

This implementation attempts to directly retrieve sensitive data fields such as ACCOUNT_NAME from the ContactsContract.Data table. In Android 17, these columns are restricted, and strict query verification causes an exception to be thrown or data to be unretrievable.

```
// NG Example: Error occurs due to direct specification of sensitive data fields
fun queryAccountNameLegacy(context: Context) {
    // *To avoid Kotlin specification constraints, string literals are directly␣
↪specified instead of constants
    val projection = arrayOf(
        ContactsContract.Data._ID,
        "account_name", // Sensitive data (ContactsContract.Data.ACCOUNT_NAME)
        "account_type"  // Sensitive data (ContactsContract.Data.ACCOUNT_TYPE)
    )

    try {
        val cursor = context.contentResolver.query(
            ContactsContract.Data.CONTENT_URI,
            projection,
            null,
            null,
            null
        )
        cursor?.use {
            // Data reading process
        }
    } catch (e: Exception) {
        // In Android 17 and later, an exception occurs
        e.printStackTrace()
    }
}
```

Android 17-Compliant Implementation (Joining and extraction using RAW_CONTACT_ID)

If it is necessary to access these sensitive data in Android 17 and later, revise the design to retrieve only RAW_CONTACT_ID from the ContactsContract.Data table and use it as a key to query the ContactsContract.RawContacts table separately.

```
// OKKotlinRAW_CONTACT_IDRawContacts
fun queryAccountNameSafely(context: Context) {
    val dataProjection = arrayOf(
        ContactsContract.Data._ID,
        ContactsContract.Data.RAW_CONTACT_ID
    )

    try {
        val dataCursor = context.contentResolver.query(
            ContactsContract.Data.CONTENT_URI,
            dataProjection,
            null,
            null,
            null
        )

        dataCursor?.use { cursor ->
            val rawContactIdIndex = cursor.getColumnIndexOrThrow(ContactsContract.
↪Data.RAW_CONTACT_ID)

            while (cursor.moveToNext()) {
                val rawContactId = cursor.getLong(rawContactIdIndex)

                val rawContactsProjection = arrayOf(
                    "account_name", // ContactsContract.RawContacts.ACCOUNT_NAME
                    "account_type"  // ContactsContract.RawContacts.ACCOUNT_TYPE
                )

                val selection = "${ContactsContract.RawContacts._ID} = ?"
                val selectionArgs = arrayOf(rawContactId.toString())

                val rawContactsCursor = context.contentResolver.query(
                    ContactsContract.RawContacts.CONTENT_URI,
                    rawContactsProjection,
                    selection,
                    selectionArgs,
                    null
                )

                rawContactsCursor?.use { rcCursor ->
                    if (rcCursor.moveToFirst()) {
                        val accountNameIndex = rcCursor.getColumnIndexOrThrow(
↪"account_name")
                        val accountName = rcCursor.getString(accountNameIndex)
                        //
                    }
                }
            }
        }
    } catch (e: Exception) {
        e.printStackTrace()
    }
```

(continues on next page)

(continued from previous page)

```
}
```

4.4 Creating/Using Services

4.4.1 Sample Code

The risks and countermeasures of using Services differ depending on how that Service is being used. You can find out which type of Service you are supposed to create through the following chart shown below. Since the secure coding best practice varies according to how the service is created, we will also explain about the implementation of the Service as well.

Table 4.4.1: Definition of service types

Type	Definition
Private Service	A service that cannot be used another application, and therefore is the safest service.
Public Service	A service that is supposed to be used by an unspecified large number of applications
Partner Service	A service that can only be used by the specific applications made by a trusted partner company.
In-house Service	A service that can only be used by other in-house applications.

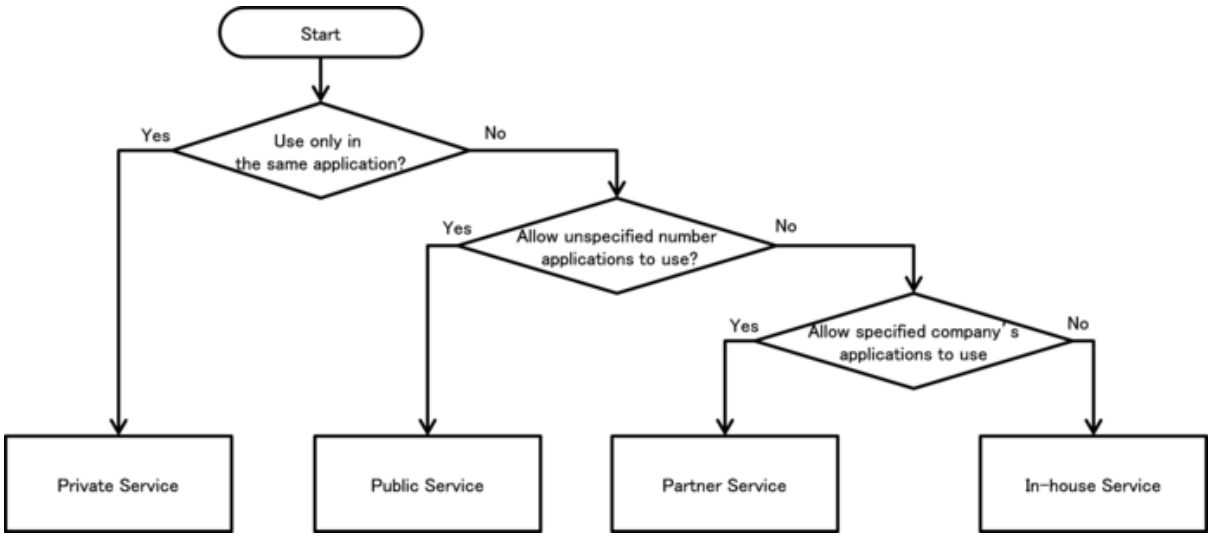


Fig. 4.4.1: Flow Figure to select Service Type

There are several implementation methods for Service, and you will select the method which matches with the type of Service that you suppose to create. The items of vertical columns in the table show the implementation methods, and these are divided into 5 types. "OK" stands for the possible combination and others show impossible/difficult combinations in the table.

Please refer to "4.4.3.2. *How to Implement Service*" and Sample code of each Service type (with * mark in a table) for detailed implementation methods of Service.

Table 4.4.2: Implementation Methods of Service

Category	Private Service	Public Service	Partner Service	In-house Service
startService type	**OK***	OK	-	OK
IntentService type	OK	**OK***	-	OK
local bind type	OK	-	-	-
Messenger bind type	OK	OK	-	**OK***
AIDL bind type	OK	OK	**OK***	OK

Sample code for each security type of Service are shown as below, by using combination of * mark in Table 4.4.2.

4.4.1.1 Creating/Using Private Services

Private Services are Services which cannot be launched by the other applications and therefore it is the safest Service.

When using Private Services that are only used within the application, as long as you use explicit Intents to the class then you do not have to worry about accidently sending it to any other application.

Sample code of how to use the startService type Service is shown below.

Points (Creating a Service):

1. Explicitly set the exported attribute to false.
2. Handle the received intent carefully and securely, even though the intent was sent from the same application.
3. Sensitive information can be sent since the requesting application is in the same application.

```
AndroidManifest.xml
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
          >

  <application
      android:icon="@drawable/ic_launcher"
      android:label="@string/app_name"
      android:allowBackup="false" >
    <activity
        android:name=".PrivateUserActivity"
        android:label="@string/app_name"
        android:exported="true" >
      <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER" />
      </intent-filter>
    </activity>

    <!-- Private Service derived from Service class -->
    <!-- *** POINT 1 *** Explicitly set the exported attribute to false. -->
    <service android:name=".PrivateStartService" android:exported="false"/>

    <!-- Private Service derived from IntentService class -->
    <!-- *** POINT 1 *** Explicitly set the exported attribute to false. -->
    <service android:name=".PrivateIntentService" android:exported="false"/>

  </application>

</manifest>
```

PrivateStartService.java

```java
/*
 * Copyright (C) 2012-2026 Japan Smartphone Security Association
 *
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 *
 *      http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */
package org.jssec.android.service.privateservice;

import android.app.Service;
import android.content.Intent;
import android.os.IBinder;
import android.widget.Toast;

public class PrivateStartService extends Service {

    // The onCreate gets called only one time when the service starts.
    @Override
    public void onCreate() {
        Toast.makeText(this, "PrivateStartService - onCreate()",
                       Toast.LENGTH_SHORT).show();
    }

    // The onStartCommand gets called each time after the startService gets called.
    @Override
    public int onStartCommand(Intent intent, int flags, int startId) {
        // *** POINT 2 *** Handle the received intent carefully and securely,
        // even though the intent was sent from the same application.
        // Omitted, since this is a sample. Please refer to
        // "3.2 Handling Input Data Carefully and Securely."
        String param = intent.getStringExtra("PARAM");
        Toast.makeText(this,
                       String.format("PrivateStartService\nReceived param: \"%s\"",
                                     param),
                       Toast.LENGTH_LONG).show();

        return Service.START_NOT_STICKY;
    }

    // The onDestroy gets called only one time when the service stops.
    @Override
    public void onDestroy() {
        Toast.makeText(this,
                       "PrivateStartService - onDestroy()",
                        Toast.LENGTH_SHORT).show();
    }

    @Override
```

(continues on next page)

(continued from previous page)

```
    public IBinder onBind(Intent intent) {
        // This service does not provide binding, so return null
        return null;
    }
}
```

Next is sample code for Activity which uses Private Service.

Points (Using a Service):

4. Use the explicit intent with class specified to call a service in the same application.
5. Sensitive information can be sent since the destination service is in the same application.
6. Handle the received result data carefully and securely, even though the data came from a service in the same application.

```
PrivateUserActivity.java
/*
 * Copyright (C) 2012-2026 Japan Smartphone Security Association
 *
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 *
 *      http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */
package org.jssec.android.service.privateservice;

import android.app.Activity;
import android.content.Intent;
import android.os.Bundle;
import android.view.View;

public class PrivateUserActivity extends Activity {

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.privateservice_activity);
    }

    // --- StartService control ---

    public void onStartServiceClick(View v) {
        // *** POINT 4 *** Use the explicit intent with class specified to call
        // a service in the same application.
        Intent intent = new Intent(this, PrivateStartService.class);

        // *** POINT 5 *** Sensitive information can be sent since the destination
        // service is in the same application.
        intent.putExtra("PARAM", "Sensitive information");
```

(continues on next page)

(continued from previous page)

```
        startService(intent);
    }

    // -- StopService control --

    public void onStopServiceClick(View v) {
        doStopService();
    }

    @Override
    public void onStop() {
        super.onStop();
        // Stop service if the service is running.
        doStopService();
    }

    private void doStopService() {
        // *** POINT 4 *** Use the explicit intent with class specified to call
        // a service in the same application.
        Intent intent = new Intent(this, PrivateStartService.class);
        stopService(intent);
    }

    // --- IntentService control ---

    public void onIntentServiceClick(View v) {
        // *** POINT 4 *** Use the explicit intent with class specified to call
        // a service in the same application.
        Intent intent = new Intent(this, PrivateIntentService.class);

        // *** POINT 5 *** Sensitive information can be sent since the destination
        // service is in the same application.
        intent.putExtra("PARAM", "Sensitive information");

        startService(intent);
    }
}
```

4.4.1.2 Creating/Using Public Services

Public Service is the Service which is supposed to be used by the unspecified large number of applications. It's necessary to pay attention that it may receive the information (Intent etc.) which was sent by Malware. In addition, since an Intent to start Service may be received by Malware, explicit Intent should be used for launching Public Service, and <intent-filter> should not be declared in Service.

Sample code of how to use the startService type Service is shown below.

Points (Creating a Service):

1. Explicitly set exported = “true” without defining the intent filter.
2. Handle the received intent carefully and securely.
3. When returning a result, do not include sensitive information.

```
AndroidManifest.xml
<?xml version="1.0" encoding="utf-8"?>
```

(continues on next page)

(continued from previous page)

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
          >

  <!-- API 28 -->
  <uses-permission android:name="android.permission.FOREGROUND_SERVICE" />

  <application
      android:icon="@drawable/ic_launcher"
      android:label="@string/app_name"
      android:allowBackup="false" >

    <!-- Most standard Service -->
    <!-- *** POINT 1 *** Explicitly set exported = "true" without defining the␣
↪intent filter. -->
    <service android:name=".PublicStartService" android:exported="true" />

    <!-- Public Service derived from IntentService class -->
    <!-- *** POINT 1 *** Explicitly set exported = "true" without defining the␣
↪intent filter. -->
    <service android:name=".PublicIntentService" android:exported="true" />

  </application>

</manifest>
```

PublicIntentService.java

```
/*
 * Copyright (C) 2012-2026 Japan Smartphone Security Association
 *
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 *
 *      http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */
package org.jssec.android.service.publicservice;

import android.app.IntentService;
import android.app.Notification;
import android.app.NotificationChannel;
import android.app.NotificationManager;
import android.content.Context;
import android.content.Intent;
import android.os.Build;
import android.widget.Toast;

public class PublicIntentService extends IntentService{

    public static final String INTENT_CHANNEL = "intent_channel";
```

(continues on next page)

(continued from previous page)

```
    /**
     * Default constructor must be provided when a service extends
     * IntentService class.
     * If it does not exist, an error occurs.
     */
    public PublicIntentService() {
        super("CreatingTypeBService");
    }

    // The onCreate gets called only one time when the Service starts.
    @Override
    public void onCreate() {
        super.onCreate();

        Toast.makeText(this,
                       this.getClass().getSimpleName() + " - onCreate()",
                       Toast.LENGTH_SHORT).show();
    }

    // The onHandleIntent gets called each time after the startService gets called.
    @Override
    protected void onHandleIntent(Intent intent) {
        if (Build.VERSION.SDK_INT >= 26) {
            Context context = getApplicationContext();
            String title = context.getString(R.string.app_name);
            NotificationChannel default_channel =
                new NotificationChannel(INTENT_CHANNEL, "Intent Channel",
                                        NotificationManager.IMPORTANCE_DEFAULT);
            NotificationManager notificationManager =
                (NotificationManager) this.getSystemService(Context.NOTIFICATION_
↪SERVICE);
            notificationManager.createNotificationChannel(default_channel);
            Notification notification =
                new Notification.Builder(context, INTENT_CHANNEL)
                    .setContentTitle(title)
                    .setSmallIcon(android.R.drawable.btn_default)
                    .setContentText("Intent Channel")
                    .setAutoCancel(true)
                    .setWhen(System.currentTimeMillis())
                    .build();
            startForeground(1, notification);
        }
        // *** POINT 2 *** Handle intent carefully and securely.
        // Since it's public service, the intent may come from malicious
        // application.
        // Omitted, since this is a sample. Please refer to
        // "3.2 Handling Input Data Carefully and Securely."
        String param = intent.getStringExtra("PARAM");
        Toast.makeText(this,
                       String.format("Recieved parameter \"%s\"", param),
                       Toast.LENGTH_LONG).show();
    }


    // The onDestroy gets called only one time when the service stops.
    @Override
```

(continues on next page)

(continued from previous page)

```
    public void onDestroy() {
        Toast.makeText(this,
                       this.getClass().getSimpleName() + " - onDestroy()",
                       Toast.LENGTH_SHORT).show();
    }

}
```

Next is sample code for Activity which uses Public Service.

Points (Using a Service):

4. Call service by Explicit Intent
5. Do not send sensitive information.
6. When receiving a result, handle the result data carefully and securely.

AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
          >

  <queries>
    <package android:name="org.jssec.android.service.publicservice" />
  </queries>

  <application
      android:icon="@drawable/ic_launcher"
      android:label="@string/app_name"
      android:allowBackup="false" >
    <activity
        android:name=".PublicUserActivity"
        android:label="@string/app_name"
        android:exported="true">
      <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER" />
      </intent-filter>
    </activity>

  </application>

</manifest>
```

PublicUserActivity.java

```
/*
 * Copyright (C) 2012-2026 Japan Smartphone Security Association
 *
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 *
 *      http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
```

(continues on next page)

(continued from previous page)

```
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */
package org.jssec.android.service.publicserviceuser;

import android.app.Activity;
import android.content.Intent;
import android.os.Build;
import android.os.Bundle;
import android.view.View;

public class PublicUserActivity extends Activity {

    // Using Service Info
    private static final String TARGET_PACKAGE =
            "org.jssec.android.service.publicservice";
    private static final String TARGET_START_CLASS =
            "org.jssec.android.service.publicservice.PublicStartService";
    private static final String TARGET_INTENT_CLASS =
            "org.jssec.android.service.publicservice.PublicIntentService";

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        setContentView(R.layout.publicservice_activity);
    }

    // --- StartService control ---

    public void onStartServiceClick(View v) {
        Intent intent = new Intent("org.jssec.android.service.publicservice.action.
↪startservice");

        // *** POINT 4 *** Call service by Explicit Intent
        intent.setClassName(TARGET_PACKAGE, TARGET_START_CLASS);

        // *** POINT 5 *** Do not send sensitive information.
        intent.putExtra("PARAM", "Not sensitive information");

        if (Build.VERSION.SDK_INT >= 26) {
            startForegroundService(intent);
        } else {
            startService(intent);
        }

        startService(intent);
        // *** POINT 6 *** When receiving a result, handle the result data
        // carefully and securely.
        // This sample code uses startService(), so receiving no result.
    }

    // --- StopService control ---

    public void onStopServiceClick(View v) {
        doStopService();
```

(continues on next page)

(continued from previous page)

```
    }

    // --- IntentService control ---

    public void onIntentServiceClick(View v) {
        Intent intent = new Intent("org.jssec.android.service.publicservice.action.
↪intentservice");

        // *** POINT 4 *** Call service by Explicit Intent
        intent.setClassName(TARGET_PACKAGE, TARGET_INTENT_CLASS);

        // *** POINT 5 *** Do not send sensitive information.
        intent.putExtra("PARAM", "Not sensitive information");

        if (Build.VERSION.SDK_INT >= 26) {
            startForegroundService(intent);
        } else {
            startService(intent);
        }
    }

    @Override
    public void onStop(){
        super.onStop();
        // Stop service if the service is running.
        doStopService();
    }

    // Stop service
    private void doStopService() {
        Intent intent = new Intent("org.jssec.android.service.publicservice.action.
↪startservice");

        // *** POINT 4 *** Call service by Explicit Intent
        intent.setClassName(TARGET_PACKAGE, TARGET_START_CLASS);

        stopService(intent);
    }
}
```

4.4.1.3 Creating/Using Partner Services

Partner Service is Service which can be used only by the particular applications. System consists of partner company's application and In house application, this is used to protect the information and features which are handled between a partner application and In house application.

Following is an example of AIDL bind type Service.

Points (Creating a Service):

1. Explicitly set exported = “true” without defining the intent filter.
2. Verify that the certificate of the requesting application has been registered in the own white list.
3. Do not (Cannot) recognize whether the requesting application is partner or not by onBind (onStartCommand, onHandleIntent).
4. Handle the received intent carefully and securely, even though the intent was sent from a partner application.

5. Return only information that is granted to be disclosed to a partner application.

In addition, refer to "5.2.1.3. *How to Verify the Hash Value of an Application's Certificate*" for how to verify the certification hash value of destination application which is specified to white list.

```
AndroidManifest.xml
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
          >

  <application
      android:icon="@drawable/ic_launcher"
      android:label="@string/app_name"
      android:allowBackup="false" >

    <!-- Service using AIDL -->
    <!-- *** POINT 1 *** Explicitly set exported = "true" without defining the␣
↪intent filter. -->
    <service
        android:name="org.jssec.android.service.partnerservice.aidl.
↪PartnerAIDLService"
        android:exported="true" />
  </application>

</manifest>
```

In this example, 2 AIDL files are to be created. One is for callback interface to give data from Service to Activity. The other one is Interface to give data from Activity to Service and to get information. In addition, package name that is described in AIDL file should be consistent with directory hierarchy in which AIDL file is created, same like package name described in java file.

```
IPartnerAIDLServiceCallback.aidl
package org.jssec.android.service.partnerservice.aidl;

interface IPartnerAIDLServiceCallback {
    /**
     * It's called when the value is changed.
     */
    void valueChanged(String info);
}
```

```
IPartnerAIDLService.aidl
package org.jssec.android.service.partnerservice.aidl;

import org.jssec.android.service.partnerservice.aidl.IExclusiveAIDLServiceCallback;

interface IPartnerAIDLService {

    /**
     * Register Callback
     */
    void registerCallback(IPartnerAIDLServiceCallback cb);

    /**
     * Get Information
     */
    String getInfo(String param);
```

(continues on next page)

(continued from previous page)

```
    /**
     * Unregister Callback
     */
    void unregisterCallback(IPartnerAIDLServiceCallback cb);
}
```

PartnerAIDLService.java

```
/*
 * Copyright (C) 2012-2026 Japan Smartphone Security Association
 *
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 *
 *      http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */
package org.jssec.android.service.partnerservice.aidl;

import org.jssec.android.shared.PkgCertWhitelists;
import org.jssec.android.shared.Utils;

import android.app.Service;
import android.content.Context;
import android.content.Intent;
import android.os.Handler;
import android.os.IBinder;
import android.os.Message;
import android.os.RemoteCallbackList;
import android.os.RemoteException;
import android.widget.Toast;

public class PartnerAIDLService extends Service {
    private static final int REPORT_MSG = 1;
    private static final int GETINFO_MSG = 2;

    // The value which this service informs to client
    private int mValue = 0;

    // *** POINT 2 *** Verify that the certificate of the requesting application
    // has been registered in the own white list.
    private static PkgCertWhitelists sWhitelists = null;
    private static void buildWhitelists(Context context) {
        boolean isdebug = Utils.isDebuggable(context);
        sWhitelists = new PkgCertWhitelists();

        // Register certificate hash value of partner application
        // "org.jssec.android.service.partnerservice.aidluser"
        sWhitelists.add("org.jssec.android.service.partnerservice.aidluser",␣
↪isdebug ?
            // Certificate hash value of  debug.keystore "androiddebugkey"
```

(continues on next page)

(continued from previous page)

```
            "0EFB7236 328348A9 89718BAD DF57F544 D5CCB4AE B9DB34BC 1E29DD26␣
↪F77C8255" :
            // Certificate hash value of keystore "partner key"
            "1F039BB5 7861C27A 3916C778 8E78CE00 690B3974 3EB8259F E2627B8D␣
↪4C0EC35A");

        // Register other partner applications in the same way
    }

    private static boolean checkPartner(Context context, String pkgname) {
        if (sWhitelists == null) buildWhitelists(context);
        return sWhitelists.test(context, pkgname);
    }

    // Object to register callback
    // Methods which RemoteCallbackList provides are thread-safe.
    private final RemoteCallbackList<IPartnerAIDLServiceCallback> mCallbacks =
        new RemoteCallbackList<IPartnerAIDLServiceCallback>();

    // Handler to send data when callback is called.
    private static class ServiceHandler extends Handler{

        private Context mContext;
        private RemoteCallbackList<IPartnerAIDLServiceCallback> mCallbacks;
        private int mValue = 0;

        public ServiceHandler(Context context,
            RemoteCallbackList<IPartnerAIDLServiceCallback> callback, int value){
            this.mContext = context;
            this.mCallbacks = callback;
            this.mValue = value;
        }

        @Override
        public void handleMessage(Message msg) {
            switch (msg.what) {
            case REPORT_MSG: {
                if(mCallbacks == null){
                    return;
                }
                // Start broadcast
                // To call back on to the registered clients, use beginBroadcast().
                // beginBroadcast() makes a copy of the currently registered
                // callback list.
                final int N = mCallbacks.beginBroadcast();
                for (int i = 0; i < N; i++) {
                    IPartnerAIDLServiceCallback target =
                            mCallbacks.getBroadcastItem(i);
                    try {
                        // *** POINT 5 *** Information that is granted to disclose
                        // to partner applications can be returned.
                        target.valueChanged("Information disclosed to partner␣
↪application (callback from Service) No." + (++mValue));

                    } catch (RemoteException e) {
                        // Callbacks are managed by RemoteCallbackList, do not
```

(continues on next page)

(continued from previous page)

```
                        // unregister callbacks here.
                        // RemoteCallbackList.kill() unregister all callbacks
                    }
                }
                // finishBroadcast() cleans up the state of a broadcast previously
                // initiated by calling beginBroadcast().
                mCallbacks.finishBroadcast();

                // Repeat after 10 seconds
                sendEmptyMessageDelayed(REPORT_MSG, 10000);
                break;
            }
            case GETINFO_MSG: {
                if(mContext != null) {
                    Toast.makeText(mContext,
                                   (String) msg.obj, Toast.LENGTH_LONG).show();
                }
                break;
            }
            default:
                super.handleMessage(msg);
                break;
            } // switch
        }
    }

    protected final ServiceHandler mHandler =
            new ServiceHandler(this, mCallbacks, mValue);

    // Interfaces defined in AIDL
    private final IPartnerAIDLService.Stub mBinder =
        new IPartnerAIDLService.Stub() {
            private boolean checkPartner() {
                Context ctx = PartnerAIDLService.this;
                if (!PartnerAIDLService.checkPartner(ctx,
                        Utils.getPackageNameFromUid(ctx, getCallingUid()))) {
                    mHandler.post(new Runnable(){
                        @Override
                        public void run(){
                            Toast.makeText(PartnerAIDLService.this,
                                "Requesting application is not partner application.
↪",
                                Toast.LENGTH_LONG).show();
                        }
                    });
                    return false;
                }
                return true;
            }
            public void registerCallback(IPartnerAIDLServiceCallback cb) {
                // *** POINT 2 *** Verify that the certificate of the requesting
                // application has been registered in the own white list.
                if (!checkPartner()) {
                    return;
                }
                if (cb != null) mCallbacks.register(cb);
```

(continues on next page)

(continued from previous page)

```
            }
            public String getInfo(String param) {
                // *** POINT 2 *** Verify that the certificate of the requesting
                // application has been registered in the own white list.
                if (!checkPartner()) {
                    return null;
                }
                // *** POINT 4 *** Handle the received intent carefully and
                // securely, even though the intent was sent from a partner
                // application
                // Omitted, since this is a sample. Please refer to
                // "3.2 Handling Input Data Carefully and Securely."
                Message msg = new Message();
                msg.what = GETINFO_MSG;
                msg.obj = String.format("Method calling from partner application.␣
↪Recieved \"%s\"", param);
                PartnerAIDLService.this.mHandler.sendMessage(msg);

                // *** POINT 5 *** Return only information that is granted to be
                // disclosed to a partner application.
                return "Information disclosed to partner application (method from␣
↪Service)";
            }

            public void unregisterCallback(IPartnerAIDLServiceCallback cb) {
                // *** POINT 2 *** Verify that the certificate of the requesting
                // application has been registered in the own white list.
                if (!checkPartner()) {
                    return;
                }

                if (cb != null) mCallbacks.unregister(cb);
            }
        };

    @Override
    public IBinder onBind(Intent intent) {
        // *** POINT 3 *** Verify that the certificate of the requesting
        // application has been registered in the own white list.
        // So requesting application must be validated in methods defined
        // in AIDL every time.
        return mBinder;
    }

    @Override
    public void onCreate() {
        Toast.makeText(this,
                       this.getClass().getSimpleName() + " - onCreate()",
                       Toast.LENGTH_SHORT).show();

        // During service is running, inform the incremented number periodically.
        mHandler.sendEmptyMessage(REPORT_MSG);
    }

    @Override
    public void onDestroy() {
```

(continues on next page)

(continued from previous page)

```
        Toast.makeText(this,
                       this.getClass().getSimpleName() + " - onDestroy()",
                       Toast.LENGTH_SHORT).show();

        // Unregister all callbacks
        mCallbacks.kill();

        mHandler.removeMessages(REPORT_MSG);
    }
}
```

PkgCertWhitelists.java

```
/*
 * Copyright (C) 2012-2026 Japan Smartphone Security Association
 *
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 *
 *      http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */

package org.jssec.android.shared;

import android.content.pm.PackageManager;
import java.util.HashMap;
import java.util.Map;
import android.content.Context;
import android.os.Build;

import static android.content.pm.PackageManager.CERT_INPUT_SHA256;

public class PkgCertWhitelists {
    private Map<String, String> mWhitelists = new HashMap<String, String>();

    public boolean add(String pkgname, String sha256) {
        if (pkgname == null) return false;
        if (sha256 == null) return false;

        sha256 = sha256.replaceAll(" ", "");
        if (sha256.length() != 64)
            return false;  // SHA-256 -> 32 bytes -> 64 chars
        sha256 = sha256.toUpperCase();
        if (sha256.replaceAll("[0-9A-F]+", "").length() != 0)
            return false;  // found non hex char

        mWhitelists.put(pkgname, sha256);
        return true;
    }
```

(continues on next page)

(continued from previous page)

```
    public boolean test(Context ctx, String pkgname) {
        // Get the correct hash value which corresponds to pkgname.
        String correctHash = mWhitelists.get(pkgname);

        // Compare the actual hash value of pkgname with the correct hash value.
        if (Build.VERSION.SDK_INT >= 28) {
            // ** if API Level >= 28, direct checking is possible
            PackageManager pm = ctx.getPackageManager();
            return pm.hasSigningCertificate(pkgname,
                                            Utils.hex2Bytes(correctHash),
                                            CERT_INPUT_SHA256);
        } else {
            // else use the facility of PkgCert
            return PkgCert.test(ctx, pkgname, correctHash);
        }
    }
}
```

PkgCert.java

```
/*
 * Copyright (C) 2012-2026 Japan Smartphone Security Association
 *
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 *
 *      http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */

package org.jssec.android.shared;

import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;

import android.content.Context;
import android.content.pm.PackageInfo;
import android.content.pm.PackageManager;
import android.content.pm.PackageManager.NameNotFoundException;
import android.content.pm.Signature;

public class PkgCert {

    public static boolean test(Context ctx, String pkgname, String correctHash) {
        if (correctHash == null) return false;
        correctHash = correctHash.replaceAll(" ", "");
        return correctHash.equals(hash(ctx, pkgname));
    }

    public static String hash(Context ctx, String pkgname) {
        if (pkgname == null) return null;
```

(continues on next page)

(continued from previous page)

```
        try {
            PackageManager pm = ctx.getPackageManager();
            PackageInfo pkginfo =
                pm.getPackageInfo(pkgname, PackageManager.GET_SIGNATURES);
            // Will not handle multiple signatures.
            if (pkginfo.signatures.length != 1) return null;
            Signature sig = pkginfo.signatures[0];
            byte[] cert = sig.toByteArray();
            byte[] sha256 = computeSha256(cert);
            return byte2hex(sha256);
        } catch (NameNotFoundException e) {
            return null;
        }
    }

    private static byte[] computeSha256(byte[] data) {
        try {
            return MessageDigest.getInstance("SHA-256").digest(data);
        } catch (NoSuchAlgorithmException e) {
            return null;
        }
    }

    private static String byte2hex(byte[] data) {
        if (data == null) return null;
        final StringBuilder hexadecimal = new StringBuilder();
        for (final byte b : data) {
            hexadecimal.append(String.format("%02X", b));
        }
        return hexadecimal.toString();
    }
}
```

Next is sample code of Activity which uses partner only Service.

Points (Using a Service):

6. Verify if the certificate of the target application has been registered in the own white list.
7. Return only information that is granted to be disclosed to a partner application.
8. Use the explicit intent to call a partner service.
9. Handle the received result data carefully and securely, even though the data came from a partner application.

PartnerAIDLUserActivity.java

```
/*
 * Copyright (C) 2012-2026 Japan Smartphone Security Association
 *
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 *
 *      http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
```

(continues on next page)

(continued from previous page)

```
 * limitations under the License.
 */
package org.jssec.android.service.partnerservice.aidluser;

import org.jssec.android.service.partnerservice.aidl.IPartnerAIDLService;
import org.jssec.android.service.partnerservice.aidl.IPartnerAIDLServiceCallback;
import org.jssec.android.shared.PkgCertWhitelists;
import org.jssec.android.shared.Utils;

import android.app.Activity;
import android.content.ComponentName;
import android.content.Context;
import android.content.Intent;
import android.content.ServiceConnection;
import android.os.Bundle;
import android.os.Handler;
import android.os.IBinder;
import android.os.Message;
import android.os.RemoteException;
import android.view.View;
import android.widget.Toast;

public class PartnerAIDLUserActivity extends Activity {

    private boolean mIsBound;
    private Context mContext;

    private final static int MGS_VALUE_CHANGED = 1;

    // *** POINT 6 *** Verify if the certificate of the target application has
    // been registered in the own white list.
    private static PkgCertWhitelists sWhitelists = null;
    private static void buildWhitelists(Context context) {
        boolean isdebug = Utils.isDebuggable(context);
        sWhitelists = new PkgCertWhitelists();

        // Register certificate hash value of partner service application
        // "org.jssec.android.service.partnerservice.aidl"
        sWhitelists.add("org.jssec.android.service.partnerservice.aidl", isdebug ?
            // Certificate hash value of debug.keystore "androiddebugkey"
            "0EFB7236 328348A9 89718BAD DF57F544 D5CCB4AE B9DB34BC 1E29DD26␣
↪F77C8255" :
            // Certificate hash value of  keystore "my company key"
            "D397D343 A5CBC10F 4EDDEB7C A10062DE 5690984F 1FB9E88B D7B3A7C2␣
↪42E142CA");

        // Register other partner service applications in the same way
    }
    private static boolean checkPartner(Context context, String pkgname) {
        if (sWhitelists == null) buildWhitelists(context);
        return sWhitelists.test(context, pkgname);
    }

    // Information about destination (requested) partner activity.
    private static final String TARGET_PACKAGE =
            "org.jssec.android.service.partnerservice.aidl";
```

(continues on next page)

(continued from previous page)

```
    private static final String TARGET_CLASS =
            "org.jssec.android.service.partnerservice.aidl.PartnerAIDLService";

    private static class ReceiveHandler extends Handler{

        private Context mContext;

        public ReceiveHandler(Context context){
            this.mContext = context;
        }

        @Override
        public void handleMessage(Message msg) {
            switch (msg.what) {
            case MGS_VALUE_CHANGED: {
                String info = (String)msg.obj;
                Toast.makeText(mContext,
                        String.format("Received \"%s\" with callback.", info),
                        Toast.LENGTH_SHORT).show();
                break;
            }
            default:
                super.handleMessage(msg);
                break;
            } // switch
        }
    }

    private final ReceiveHandler mHandler = new ReceiveHandler(this);

    // Interfaces defined in AIDL. Receive notice from service
    private final IPartnerAIDLServiceCallback.Stub mCallback =
        new IPartnerAIDLServiceCallback.Stub() {
            @Override
            public void valueChanged(String info) throws RemoteException {
                Message msg = mHandler.obtainMessage(MGS_VALUE_CHANGED, info);
                mHandler.sendMessage(msg);
            }
        };

    // Interfaces defined in AIDL. Inform service.
    private IPartnerAIDLService mService = null;

    // Connection used to connect with service. This is necessary when service is
    // implemented with bindService().
    private ServiceConnection mConnection = new ServiceConnection() {

            // This is called when the connection with the service has been
            // established.
            @Override
            public void onServiceConnected(ComponentName className,
                                           IBinder service) {
                mService = IPartnerAIDLService.Stub.asInterface(service);

                try{
                    // connect to service
```

(continues on next page)

(continued from previous page)

```
                mService.registerCallback(mCallback);

            }catch(RemoteException e){
                // service stopped abnormally
            }

            Toast.makeText(mContext,
                           "Connected to service",
                           Toast.LENGTH_SHORT).show();
        }

        // This is called when the service stopped abnormally and connection
        // is disconnected.
        @Override
        public void onServiceDisconnected(ComponentName className) {
            Toast.makeText(mContext,
                           "Disconnected from service",
                           Toast.LENGTH_SHORT).show();
        }
    };

@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);

    setContentView(R.layout.partnerservice_activity);

    mContext = this;
}

// -- StartService control --
public void onStartServiceClick(View v) {
    // Start bindService
    doBindService();
}

// -- GetInfo control --
public void onGetInfoClick(View v) {
    getServiceinfo();
}

// -- StopService control --
public void onStopServiceClick(View v) {
    doUnbindService();
}

@Override
public void onDestroy() {
    super.onDestroy();
    doUnbindService();
}

/**
 * Connect to service
 */
private void doBindService() {
```

(continues on next page)

(continued from previous page)

```
        if (!mIsBound){
            // *** POINT 6 *** Verify if the certificate of the target application
            // has been registered in the own white list.
            if (!checkPartner(this, TARGET_PACKAGE)) {
                Toast.makeText(this,
                    "Destination(Requested) sevice application is not registered␣
↪in white list.", Toast.LENGTH_LONG).show();
                return;
            }

            Intent intent = new Intent();

            // *** POINT 7 *** Return only information that is granted to be
            // disclosed to a partner application.
            intent.putExtra("PARAM",
                            "Information disclosed to partner application");

            // *** POINT 8 *** Use the explicit intent to call a partner service.
            intent.setClassName(TARGET_PACKAGE, TARGET_CLASS);

            bindService(intent, mConnection, Context.BIND_AUTO_CREATE);
            mIsBound = true;
        }
    }

    /**
     * Disconnect service
     */
    private void doUnbindService() {
        if (mIsBound) {
            // Unregister callbacks which have been registered.
            if(mService != null){
                try{
                    mService.unregisterCallback(mCallback);
                }catch(RemoteException e){
                    // Service stopped abnormally
                    // Omitted, since it' s sample.
                }
            }

            unbindService(mConnection);

            Intent intent = new Intent();

            // *** POINT 8 *** Use the explicit intent to call a partner service.
            intent.setClassName(TARGET_PACKAGE, TARGET_CLASS);

            stopService(intent);

            mIsBound = false;
        }
    }

    /**
     * Get information from service
     */
```

(continues on next page)

(continued from previous page)

```
    void getServiceinfo() {
        if (mIsBound && mService != null) {
            String info = null;

            try {
                // *** POINT 7 *** Return only information that is granted to be
                // disclosed to a partner application.
                info = mService.getInfo("Information disclosed to partner␣
↪application (method from activity)");
            } catch (RemoteException e) {
                e.printStackTrace();
            }
            // *** POINT 9 *** Handle the received result data carefully and
            // securely, even though the data came from a partner application.
            // Omitted, since this is a sample. Please refer to
            // "3.2 Handling Input Data Carefully and Securely."
            Toast.makeText(mContext,
                           String.format("Received \"%s\" from service.", info),
                           Toast.LENGTH_SHORT).show();
        }
    }
}
```

PkgCertWhitelists.java

```
/*
 * Copyright (C) 2012-2026 Japan Smartphone Security Association
 *
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 *
 *      http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */

package org.jssec.android.shared;

import android.content.pm.PackageManager;
import java.util.HashMap;
import java.util.Map;
import android.content.Context;
import android.os.Build;

import static android.content.pm.PackageManager.CERT_INPUT_SHA256;

public class PkgCertWhitelists {
    private Map<String, String> mWhitelists = new HashMap<String, String>();

    public boolean add(String pkgname, String sha256) {
        if (pkgname == null) return false;
        if (sha256 == null) return false;
```

(continues on next page)

(continued from previous page)

```
        sha256 = sha256.replaceAll(" ", "");
        if (sha256.length() != 64)
            return false;  // SHA-256 -> 32 bytes -> 64 chars
        sha256 = sha256.toUpperCase();
        if (sha256.replaceAll("[0-9A-F]+", "").length() != 0)
            return false;  // found non hex char

        mWhitelists.put(pkgname, sha256);
        return true;
    }

    public boolean test(Context ctx, String pkgname) {
        // Get the correct hash value which corresponds to pkgname.
        String correctHash = mWhitelists.get(pkgname);

        // Compare the actual hash value of pkgname with the correct hash value.
        if (Build.VERSION.SDK_INT >= 28) {
            // ** if API Level >= 28, direct checking is possible
            PackageManager pm = ctx.getPackageManager();
            return pm.hasSigningCertificate(pkgname,
                                            Utils.hex2Bytes(correctHash),
                                            CERT_INPUT_SHA256);
        } else {
            // else use the facility of PkgCert
            return PkgCert.test(ctx, pkgname, correctHash);
        }
    }
}
```

PkgCert.java

```
/*
 * Copyright (C) 2012-2026 Japan Smartphone Security Association
 *
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 *
 *      http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */

package org.jssec.android.shared;

import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;

import android.content.Context;
import android.content.pm.PackageInfo;
import android.content.pm.PackageManager;
import android.content.pm.PackageManager.NameNotFoundException;
```

(continues on next page)

(continued from previous page)

```
import android.content.pm.Signature;

public class PkgCert {

    public static boolean test(Context ctx, String pkgname, String correctHash) {
        if (correctHash == null) return false;
        correctHash = correctHash.replaceAll(" ", "");
        return correctHash.equals(hash(ctx, pkgname));
    }

    public static String hash(Context ctx, String pkgname) {
        if (pkgname == null) return null;
        try {
            PackageManager pm = ctx.getPackageManager();
            PackageInfo pkginfo =
                pm.getPackageInfo(pkgname, PackageManager.GET_SIGNATURES);
            // Will not handle multiple signatures.
            if (pkginfo.signatures.length != 1) return null;
            Signature sig = pkginfo.signatures[0];
            byte[] cert = sig.toByteArray();
            byte[] sha256 = computeSha256(cert);
            return byte2hex(sha256);
        } catch (NameNotFoundException e) {
            return null;
        }
    }

    private static byte[] computeSha256(byte[] data) {
        try {
            return MessageDigest.getInstance("SHA-256").digest(data);
        } catch (NoSuchAlgorithmException e) {
            return null;
        }
    }

    private static String byte2hex(byte[] data) {
        if (data == null) return null;
        final StringBuilder hexadecimal = new StringBuilder();
        for (final byte b : data) {
            hexadecimal.append(String.format("%02X", b));
        }
        return hexadecimal.toString();
    }
}
```

4.4.1.4 Creating/Using In-house Services

In-house Services are the Services which are prohibited to be used by applications other than in-house applications. They are used in applications developed internally that want to securely share information and functionality.

Following is an example which uses Messenger bind type Service.

Points (Creating a Service):

1. Define an in-house signature permission.
2. Require the in-house signature permission.

3. Explicitly set exported = “true” without defining the intent filter.
4. Verify that the in-house signature permission is defined by an in-house application.
5. Handle the received intent carefully and securely, even though the intent was sent from an in-house application.
6. Sensitive information can be returned since the requesting application is in-house.
7. When exporting an APK, sign the APK with the same developer key as the requesting application.

AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
          >

  <!-- *** POINT 1 *** Define an in-house signature permission -->
  <permission
      android:name="org.jssec.android.service.inhouseservice.messenger.MY_
↪PERMISSION"
      android:protectionLevel="signature" />

  <application
      android:icon="@drawable/ic_launcher"
      android:label="@string/app_name"
      android:allowBackup="false" >

    <!-- Service using Messenger -->
    <!-- *** POINT 2 *** Require the in-house signature permission -->
    <!-- *** POINT 3 *** Explicitly set exported = "true" without defining the␣
↪intent filter. -->
    <service
        android:name="org.jssec.android.service.inhouseservice.messenger.
↪InhouseMessengerService"
        android:exported="true"
        android:permission="org.jssec.android.service.inhouseservice.messenger.MY_
↪PERMISSION" />
  </application>

</manifest>
```

InhouseMessengerService.java

```
/*
 * Copyright (C) 2012-2026 Japan Smartphone Security Association
 *
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 *
 *      http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */

package org.jssec.android.service.inhouseservice.messenger;
```

(continues on next page)

(continued from previous page)

```
import org.jssec.android.shared.SigPerm;
import org.jssec.android.shared.Utils;

import java.lang.reflect.Array;
import java.util.ArrayList;
import java.util.Iterator;

import android.app.Service;
import android.content.Context;
import android.content.Intent;
import android.os.Bundle;
import android.os.Handler;
import android.os.IBinder;
import android.os.Message;
import android.os.Messenger;
import android.os.RemoteException;
import android.widget.Toast;

public class InhouseMessengerService extends Service{
    // In-house signature permission
    private static final String MY_PERMISSION =
        "org.jssec.android.service.inhouseservice.messenger.MY_PERMISSION";

    // In-house certificate hash value
    private static String sMyCertHash = null;
    private static String myCertHash(Context context) {
        if (sMyCertHash == null) {
            if (Utils.isDebuggable(context)) {
                // Certificate hash value of debug.keystore "androiddebugkey"
                sMyCertHash = "0EFB7236 328348A9 89718BAD DF57F544 D5CCB4AE␣
↪B9DB34BC 1E29DD26 F77C8255";
            } else {
                // Certificate hash value of keystore "my company key"
                sMyCertHash = "D397D343 A5CBC10F 4EDDEB7C A10062DE 5690984F␣
↪1FB9E88B D7B3A7C2 42E142CA";
            }
        }
        return sMyCertHash;
    }


    // Manage clients(destinations of sending data) in a list
    private ArrayList<Messenger> mClients = new ArrayList<Messenger>();

    // Messenger used when service receive data from client
    private final Messenger mMessenger =
            new Messenger(new ServiceSideHandler(mClients));

    // Handler which handles message received from client
    private static class ServiceSideHandler extends Handler{

        private ArrayList<Messenger> mClients;

        public ServiceSideHandler(ArrayList<Messenger> clients){
            mClients = clients;
        }
```

(continues on next page)

(continued from previous page)

```
    @Override
    public void handleMessage(Message msg){
        switch(msg.what){
        case CommonValue.MSG_REGISTER_CLIENT:
            // Add messenger received from client
            mClients.add(msg.replyTo);
            break;
        case CommonValue.MSG_UNREGISTER_CLIENT:
            mClients.remove(msg.replyTo);
            break;
        case CommonValue.MSG_SET_VALUE:
            // Send data to client
            sendMessageToClients(mClients);
            break;
        default:
            super.handleMessage(msg);
            break;
        }
    }
}

/**
 * Send data to client
 */
private static void sendMessageToClients(ArrayList<Messenger> mClients){

    // *** POINT 6 *** Sensitive information can be returned since the
    // requesting application is in-house.
    String sendValue = "Sensitive information (from Service)";

    // Send data to the registered client one by one.
    // Use iterator to send all clients even though clients are removed in the
    // loop process.
    Iterator<Messenger> ite = mClients.iterator();
    while(ite.hasNext()){
        try {
            Message sendMsg =
                Message.obtain(null, CommonValue.MSG_SET_VALUE, null);

            Bundle data = new Bundle();
            data.putString("key", sendValue);
            sendMsg.setData(data);

            Messenger next = ite.next();
            next.send(sendMsg);

        } catch (RemoteException e) {
            // If client does not exits, remove it from a list.
            ite.remove();
        }
    }
}

@Override
public IBinder onBind(Intent intent) {
```

(continues on next page)

(continued from previous page)

```
        // *** POINT 4 *** Verify that the in-house signature permission is
        // defined by an in-house application.
        if (!SigPerm.test(this, MY_PERMISSION, myCertHash(this))) {
            Toast.makeText(this,"In-house defined signature permission is not␣
↪defined by in-house application.", Toast.LENGTH_LONG).show();
            return null;
        }

        // *** POINT 5 *** Handle the received intent carefully and securely,
        // even though the intent was sent from an in-house application.
        // Omitted, since this is a sample. Please refer to
        // "3.2 Handling Input Data Carefully and Securely."
        String param = intent.getStringExtra("PARAM");
        Toast.makeText(this,
                       String.format("Received parameter \"%s\".", param),
                       Toast.LENGTH_LONG).show();

        return mMessenger.getBinder();
    }

    @Override
    public void onCreate() {
        Toast.makeText(this, "Service - onCreate()", Toast.LENGTH_SHORT).show();
    }

    @Override
    public void onDestroy() {
        Toast.makeText(this, "Service - onDestroy()", Toast.LENGTH_SHORT).show();
    }
}
```

SigPerm.java

```
/*
 * Copyright (C) 2012-2026 Japan Smartphone Security Association
 *
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 *
 *      http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */

package org.jssec.android.shared;

import android.content.Context;
import android.content.pm.PackageManager;
import android.content.pm.PackageManager.NameNotFoundException;
import android.content.pm.PermissionInfo;
import android.os.Build;
```

(continues on next page)

(continued from previous page)

```
import static android.content.pm.PackageManager.CERT_INPUT_SHA256;

public class SigPerm {

    public static boolean test(Context ctx, String sigPermName,
                               String correctHash) {
        if (correctHash == null) return false;
        correctHash = correctHash.replaceAll(" ", "");
        try {
            // Get the package name of the application which declares a permission
            // named sigPermName.
            PackageManager pm = ctx.getPackageManager();
            PermissionInfo pi =
                pm.getPermissionInfo(sigPermName, PackageManager.GET_META_DATA);
            String pkgname = pi.packageName;
            // Fail if the permission named sigPermName is not a Signature
            // Permission
            if (pi.protectionLevel != PermissionInfo.PROTECTION_SIGNATURE)
                return false;

            // Compare the actual hash value of pkgname with the correct hash
            // value.
            if (Build.VERSION.SDK_INT >= 28) {
                // ** if API Level >= 28, direct check is possible
                return pm.hasSigningCertificate(pkgname,
                                                Utils.hex2Bytes(correctHash),
                                                CERT_INPUT_SHA256);
            } else {
                // else(API Level < 28) use the facility of PkgCert
                return correctHash.equals(PkgCert.hash(ctx, pkgname));
            }

        } catch (NameNotFoundException e) {
            return false;
        }
    }
}
```

PkgCert.java

```
/*
 * Copyright (C) 2012-2026 Japan Smartphone Security Association
 *
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 *
 *      http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */
```

(continues on next page)

(continued from previous page)

```
package org.jssec.android.shared;

import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;

import android.content.Context;
import android.content.pm.PackageInfo;
import android.content.pm.PackageManager;
import android.content.pm.PackageManager.NameNotFoundException;
import android.content.pm.Signature;

public class PkgCert {

    public static boolean test(Context ctx, String pkgname, String correctHash) {
        if (correctHash == null) return false;
        correctHash = correctHash.replaceAll(" ", "");
        return correctHash.equals(hash(ctx, pkgname));
    }

    public static String hash(Context ctx, String pkgname) {
        if (pkgname == null) return null;
        try {
            PackageManager pm = ctx.getPackageManager();
            PackageInfo pkginfo =
                pm.getPackageInfo(pkgname, PackageManager.GET_SIGNATURES);
            // Will not handle multiple signatures.
            if (pkginfo.signatures.length != 1) return null;
            Signature sig = pkginfo.signatures[0];
            byte[] cert = sig.toByteArray();
            byte[] sha256 = computeSha256(cert);
            return byte2hex(sha256);
        } catch (NameNotFoundException e) {
            return null;
        }
    }

    private static byte[] computeSha256(byte[] data) {
        try {
            return MessageDigest.getInstance("SHA-256").digest(data);
        } catch (NoSuchAlgorithmException e) {
            return null;
        }
    }

    private static String byte2hex(byte[] data) {
        if (data == null) return null;
        final StringBuilder hexadecimal = new StringBuilder();
        for (final byte b : data) {
            hexadecimal.append(String.format("%02X", b));
        }
        return hexadecimal.toString();
    }
}
```

*** Point 7 *** When exporting an APK, sign the APK with the same developer key as the requesting application.

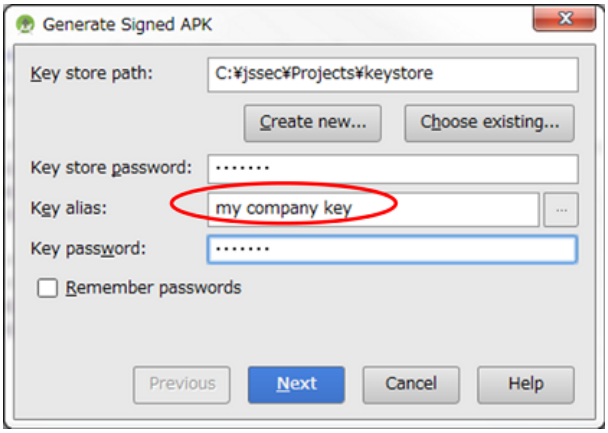


Fig. 4.4.2: Sign the APK with the same developer key as the requesting application

Next is the sample code of Activity which uses in house only Service.

Points (Using a Service):

8. Declare to use the in-house signature permission.
9. Verify that the in-house signature permission is defined by an in-house application.
10. Verify that the destination application is signed with the in-house certificate.
11. Sensitive information can be sent since the destination application is in-house.
12. Use the explicit intent to call an in-house service.
13. Handle the received result data carefully and securely, even though the data came from an in-house application.
14. When exporting an APK, sign the APK with the same developer key as the destination application.

```
AndroidManifest.xml
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
          >

  <queries>
    <package android:name="org.jssec.android.service.inhouseservice.messenger" />
  </queries>

  <!-- *** POINT 8 *** Declare to use the in-house signature permission.  -->
  <uses-permission
      android:name="org.jssec.android.service.inhouseservice.messenger.MY_
↪PERMISSION" />

  <application
      android:icon="@drawable/ic_launcher"
      android:label="@string/app_name"
      android:allowBackup="false" >
    <activity
        android:name="org.jssec.android.service.inhouseservice.messengeruser.
↪InhouseMessengerUserActivity"
        android:label="@string/app_name"
        android:exported="true" >
      <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER" />
      </intent-filter>
```

(continues on next page)

(continued from previous page)

```
    </activity>
  </application>

</manifest>
```

InhouseMessengerUserActivity.java

```
/*
 * Copyright (C) 2012-2026 Japan Smartphone Security Association
 *
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 *
 *      http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */

package org.jssec.android.service.inhouseservice.messengeruser;

import org.jssec.android.shared.PkgCert;
import org.jssec.android.shared.SigPerm;
import org.jssec.android.shared.Utils;

import android.app.Activity;
import android.content.ComponentName;
import android.content.Context;
import android.content.Intent;
import android.content.ServiceConnection;
import android.os.Bundle;
import android.os.Handler;
import android.os.IBinder;
import android.os.Message;
import android.os.Messenger;
import android.os.RemoteException;
import android.view.View;
import android.widget.Toast;

public class InhouseMessengerUserActivity extends Activity {

    private boolean mIsBound;
    private Context mContext;

    // Destination (Requested) service application information
    private static final String TARGET_PACKAGE =
        "org.jssec.android.service.inhouseservice.messenger";
    private static final String TARGET_CLASS =
        "org.jssec.android.service.inhouseservice.messenger.InhouseMessengerService
↪";

    // In-house signature permission
    private static final String MY_PERMISSION =
```

(continues on next page)

(continued from previous page)

```
        "org.jssec.android.service.inhouseservice.messenger.MY_PERMISSION";

    // In-house certificate hash value
    private static String sMyCertHash = null;
    private static String myCertHash(Context context) {
        if (sMyCertHash == null) {
            if (Utils.isDebuggable(context)) {
                // Certificate hash value of debug.keystore "androiddebugkey"
                sMyCertHash = "0EFB7236 328348A9 89718BAD DF57F544 D5CCB4AE␣
↪B9DB34BC 1E29DD26 F77C8255";
            } else {
                // Certificate hash value of keystore "my company key"
                sMyCertHash = "D397D343 A5CBC10F 4EDDEB7C A10062DE 5690984F␣
↪1FB9E88B D7B3A7C2 42E142CA";
            }
        }
        return sMyCertHash;
    }

    // Messenger used when this application receives data from service.
    private Messenger mServiceMessenger = null;

    // Messenger used when this application sends data to service.
    private final Messenger mActivityMessenger =
            new Messenger(new ActivitySideHandler());

    // Handler which handles message received from service
    private class ActivitySideHandler extends Handler {
        @Override
        public void handleMessage(Message msg) {
            switch (msg.what) {
            case CommonValue.MSG_SET_VALUE:
                Bundle data = msg.getData();
                String info = data.getString("key");
                // *** POINT 13 *** Handle the received result data carefully and
                // securely, even though the data came from an in-house application
                // Omitted, since this is a sample. Please refer to
                // "3.2 Handling Input Data Carefully and Securely."
                Toast.makeText(mContext,
                        String.format("Received \"%s\" from service.", info),
                        Toast.LENGTH_SHORT).show();
                break;
            default:
                super.handleMessage(msg);
            }
        }
    }

    // Connection used to connect with service. This is necessary when service is
    // implemented with bindService().
    private ServiceConnection mConnection = new ServiceConnection() {

            // This is called when the connection with the service has been
            // established.
            @Override
            public void onServiceConnected(ComponentName className,
```

(continues on next page)

(continued from previous page)

```
                                        IBinder service) {
                mServiceMessenger = new Messenger(service);
                Toast.makeText(mContext,
                               "Connect to service",
                               Toast.LENGTH_SHORT).show();

                try {
                    // Send own messenger to service
                    Message msg =
                        Message.obtain(null, CommonValue.MSG_REGISTER_CLIENT);
                    msg.replyTo = mActivityMessenger;
                    mServiceMessenger.send(msg);
                } catch (RemoteException e) {
                    // Service stopped abnormally
                }
            }

            // This is called when the service stopped abnormally and connection
            // is disconnected.
            @Override
            public void onServiceDisconnected(ComponentName className) {
                mServiceMessenger = null;
                Toast.makeText(mContext,
                               "Disconnected from service",
                               Toast.LENGTH_SHORT).show();
            }
        };

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        setContentView(R.layout.inhouseservice_activity);

        mContext = this;
    }

    // --- StartService control ---
    public void onStartServiceClick(View v) {
        // Start bindService
        doBindService();
    }

    // -- GetInfo control --
    public void onGetInfoClick(View v) {
        getServiceinfo();
    }

    // -- StopService control --
    public void onStopServiceClick(View v) {
        doUnbindService();
    }

    @Override
    protected void onDestroy() {
        super.onDestroy();
```

(continues on next page)

(continued from previous page)

```
        doUnbindService();
    }

    /**
     * Connect to service
     */
    void doBindService() {
        if (!mIsBound){
            // *** POINT 9 *** Verify that the in-house signature permission is
            // defined by an in-house application.
            if (!SigPerm.test(this, MY_PERMISSION, myCertHash(this))) {
                Toast.makeText(this, "In-house defined signature permission is not␣
↪defined by in-house application.", Toast.LENGTH_LONG).show();
                return;
            }

            // *** POINT 10 *** Verify that the destination application is signed
            // with the in-house certificate.
            if (!PkgCert.test(this, TARGET_PACKAGE, myCertHash(this))) {
                Toast.makeText(this, "Destination(Requested) service application␣
↪is not in-house application.", Toast.LENGTH_LONG).show();
                return;
            }

            Intent intent = new Intent();

            // *** POINT 11 *** Sensitive information can be sent since the
            // destination application is in-house one.
            intent.putExtra("PARAM", "Sensitive information");

            // *** POINT 12 *** Use the explicit intent to call an in-house
            // service.
            intent.setClassName(TARGET_PACKAGE, TARGET_CLASS);

            bindService(intent, mConnection, Context.BIND_AUTO_CREATE);
            mIsBound = true;
        }
    }

    /**
     * Disconnect service
     */
    void doUnbindService() {
        if (mIsBound) {
            unbindService(mConnection);
            mIsBound = false;
        }
    }

    /**
     * Get information from service
     */
    void getServiceinfo() {
        if (mServiceMessenger != null) {
            try {
                // Request sending information
```

(continues on next page)

(continued from previous page)

```
                Message msg = Message.obtain(null, CommonValue.MSG_SET_VALUE);
                mServiceMessenger.send(msg);
            } catch (RemoteException e) {
                // Service stopped abnormally
            }
        }
    }
}
```

SigPerm.java

```
/*
 * Copyright (C) 2012-2026 Japan Smartphone Security Association
 *
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 *
 *      http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */

package org.jssec.android.shared;

import android.content.Context;
import android.content.pm.PackageManager;
import android.content.pm.PackageManager.NameNotFoundException;
import android.content.pm.PermissionInfo;
import android.os.Build;

import static android.content.pm.PackageManager.CERT_INPUT_SHA256;

public class SigPerm {

    public static boolean test(Context ctx, String sigPermName,
                               String correctHash) {
        if (correctHash == null) return false;
        correctHash = correctHash.replaceAll(" ", "");
        try {
            // Get the package name of the application which declares a permission
            // named sigPermName.
            PackageManager pm = ctx.getPackageManager();
            PermissionInfo pi =
                pm.getPermissionInfo(sigPermName, PackageManager.GET_META_DATA);
            String pkgname = pi.packageName;
            // Fail if the permission named sigPermName is not a Signature
            // Permission
            if (pi.protectionLevel != PermissionInfo.PROTECTION_SIGNATURE)
                return false;

            // Compare the actual hash value of pkgname with the correct hash
            // value.
```

(continues on next page)

(continued from previous page)

```
            if (Build.VERSION.SDK_INT >= 28) {
                // ** if API Level >= 28, direct check is possible
                return pm.hasSigningCertificate(pkgname,
                                                Utils.hex2Bytes(correctHash),
                                                CERT_INPUT_SHA256);
            } else {
                // else(API Level < 28) use the facility of PkgCert
                return correctHash.equals(PkgCert.hash(ctx, pkgname));
            }

        } catch (NameNotFoundException e) {
            return false;
        }
    }
}
```

PkgCert.java

```
/*
 * Copyright (C) 2012-2026 Japan Smartphone Security Association
 *
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 *
 *      http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */

package org.jssec.android.shared;

import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;

import android.content.Context;
import android.content.pm.PackageInfo;
import android.content.pm.PackageManager;
import android.content.pm.PackageManager.NameNotFoundException;
import android.content.pm.Signature;

public class PkgCert {

    public static boolean test(Context ctx, String pkgname, String correctHash) {
        if (correctHash == null) return false;
        correctHash = correctHash.replaceAll(" ", "");
        return correctHash.equals(hash(ctx, pkgname));
    }

    public static String hash(Context ctx, String pkgname) {
        if (pkgname == null) return null;
        try {
            PackageManager pm = ctx.getPackageManager();
```

(continues on next page)

(continued from previous page)

```
            PackageInfo pkginfo =
                pm.getPackageInfo(pkgname, PackageManager.GET_SIGNATURES);
            // Will not handle multiple signatures.
            if (pkginfo.signatures.length != 1) return null;
            Signature sig = pkginfo.signatures[0];
            byte[] cert = sig.toByteArray();
            byte[] sha256 = computeSha256(cert);
            return byte2hex(sha256);
        } catch (NameNotFoundException e) {
            return null;
        }
    }

    private static byte[] computeSha256(byte[] data) {
        try {
            return MessageDigest.getInstance("SHA-256").digest(data);
        } catch (NoSuchAlgorithmException e) {
            return null;
        }
    }

    private static String byte2hex(byte[] data) {
        if (data == null) return null;
        final StringBuilder hexadecimal = new StringBuilder();
        for (final byte b : data) {
            hexadecimal.append(String.format("%02X", b));
        }
        return hexadecimal.toString();
    }
}
```

*** Point14 *** When exporting an APK, sign the APK with the same developer key as the destination application.

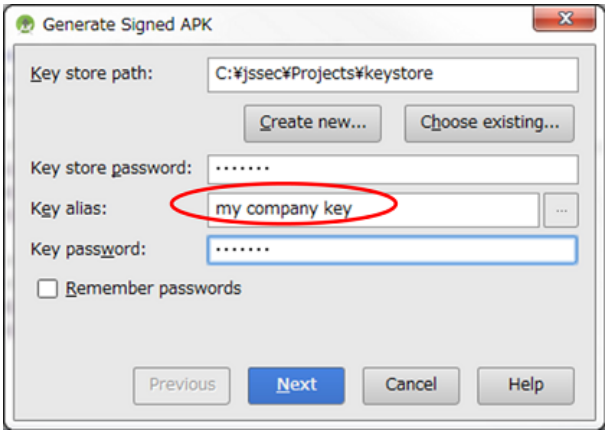


Fig. 4.4.3: Sign the APK with the same developer key as the destination application

4.4.2 Rule Book

Implementing or using service, follow the rules below.

1. *Service that Is Used Only in an application, Must Be Set as Private (Required)*
2. *Handle the Received Data Carefully and Securely (Required)*

3. *Use the In-house Defined Signature Permission after Verifying If it's Defined by an In-house Application (Required)*
4. *Do Not Determine Whether the Service Provides its Functions, in onCreate (Required)*
5. *When Returning a Result Information, Pay Attention the Result Information Leakage from the Destination Application (Required)*
6. *Use the Explicit Intent if the Destination Service Is fixed (Required)*
7. *Verify the Destination Service If Linking with the Other Company's Application (Required)*
8. *When Providing an Asset Secondarily, the Asset should be protected with the Same Level Protection (Required)*
9. *Sensitive Information Should Not Be Sent As Much As Possible (Recommended)*

4.4.2.1 Service that Is Used Only in an application, Must Be Set as Private (Required)

Service that is used only in an application (or in same UID) must be set as Private. It avoids the application from receiving Intents from other applications unexpectedly and eventually prevents from damages such as application functions are used or application behavior becomes abnormal.

All you have to do in implementation is set exported attribute false when defining Service in AndroidManifest.xml.

```
AndroidManifest.xml
    <!-- Private Service derived from Service class -->
    <!-- *** 4.4.1.1 - POINT 1 *** Explicitly set the exported attribute to false.␣
↪-->
    <service android:name=".PrivateStartService" android:exported="false"/>
```

In addition, this is a rare case, but do not set Intent Filter when service is used only within the application. The reason is that, due to the characteristics of Intent Filter, public service in other application may be called unexpectedly though you intend to call Private Service within the application.

```
AndroidManifest.xml(Not recommended)
    <!-- Private Service derived from Service class -->
    <!-- *** 4.4.1.1 - POINT 1 *** Explicitly set the exported attribute to false.␣
↪-->
    <service android:name=".PrivateStartService" android:exported="false">
        <intent-filter>
            <action android:name="org.jssec.android.service.OPEN" />
        </intent-filter>
    </service>
```

See "4.4.3.1. *Combination of Exported Attribute and Intent-filter Setting (In the Case of Service)*".

4.4.2.2 Handle the Received Data Carefully and Securely (Required)

Same like Activity, In case of Service, when processing a received Intent data, the first thing you should do is input validation. Also in Service user side, it's necessary to verify the safety of result information from Service. Please refer to "4.1.2.5. *Handling the Received Intent Carefully and Securely (Required)*" and "4.1.2.9. *Handle the Returned Data from a Requested Activity Carefully and Securely (Required)*."

In Service, you should also implement calling method and exchanging data by Message carefully.

Please refer to "3.2. *Handling Input Data Carefully and Securely*"

4.4.2.3 Use the In-house Defined Signature Permission after Verifying If it's Defined by an In-house Application (Required)

Make sure to protect your in-house Services by defining in-house signature permission when creating the Service. Since defining a permission in the AndroidManifest.xml file or declaring a permission request does not provide adequate security, please be sure to refer to "5.2.1.2. *How to Communicate Between In-house Applications with In-house-defined Signature Permission*."

4.4.2.4 Do Not Determine Whether the Service Provides its Functions, in onCreate (Required)

Security checks such as Intent parameter verification or in-house-defined Signature Permission verification should not be included in onCreate, because when receiving new request during Service is running, process of onCreate is not executed. So, when implementing Service which is started by startService, judgment should be executed by onStartCommand (In case of using IntentService, judgment should be executed by onHandleIntent.) It's also same in the case when implementing Service which is started by bindService, judgment should be executed by onBind.

4.4.2.5 When Returning a Result Information, Pay Attention the Result Information Leakage from the Destination Application (Required)

Depends on types of Service, the reliability of result information destination application (callback receiver side/ Message destination) are different. Need to consider seriously about the information leakage considering the possibility that the destination may be Malware.

See, Activity "4.1.2.7. *When Returning a Result, Pay Attention to the Possibility of Information Leakage of that Result from the Destination Application (Required)*", for details.

4.4.2.6 Use the Explicit Intent if the Destination Service Is fixed (Required)

When using a Service by implicit Intents, in case the definition of Intent Filter is same, Intent is sent to the Service which was installed earlier. If Malware with the same Intent Filter defined intentionally was installed earlier, Intent is sent to Malware and information leakage occurs. On the other hand, when using a Service by explicit Intents, only the intended Service will receive the Intent so this is much safer.

There are some other points which should be considered, please refer to "4.1.2.8. *Use the explicit Intents if the destination Activity is predetermined. (Required)*."

4.4.2.7 Verify the Destination Service If Linking with the Other Company's Application (Required)

Be sure to sure a whitelist when linking with another company's application. You can do this by saving a copy of the company's certificate hash inside your application and checking it with the certificate hash of the destination application. This will prevent a malicious application from being able to spoof Intents. Please refer to sample code section "4.4.1.3. *Creating/Using Partner Services*" for the concrete implementation method.

4.4.2.8 When Providing an Asset Secondarily, the Asset should be protected with the Same Level Protection (Required)

When an information or function asset, which is protected by permission, is provided to another application secondhand, you need to make sure that it has the same required permissions needed to access the asset. In the Android OS permission security model, only an application that has been granted proper permissions can directly access a protected asset. However, there is a loophole because an application with permissions to an asset can act as a proxy and allow access to an unprivileged application. Substantially this is the same as re-delegating permission so it is referred to as the "Permission Re-delegation" problem. Please refer to "5.2.3.4. *Permission Re-delegation Problem*."

4.4.2.9 Sensitive Information Should Not Be Sent As Much As Possible (Recommended)

You should not send sensitive information to untrusted parties.

You need to consider the risk of information leakage when exchanging sensitive information with a Service. You must assume that all data in Intents sent to a Public Service can be obtained by a malicious third party. In addition, there is a variety of risks of information leakage when sending Intents to Partner or In-house Services as well depending on the implementation.

Not sending sensitive data in the first place is the only perfect solution to prevent information leakage therefore you should limit the amount of sensitive information being sent as much as possible. When it is necessary to send sensitive information, the best practice is to only send to a trusted Service and to make sure the information cannot be leaked through LogCat.

4.4.3 Advanced Topics

4.4.3.1 Combination of Exported Attribute and Intent-filter Setting (In the Case of Service)

We have explained how to implement the four types of Services in this guidebook: Private Services, Public Services, Partner Services, and In-house Services. The various combinations of permitted settings for each type of exported attribute defined in the AndroidManifest.xml file and the intent-filter elements are defined in the table below. Please verify the compatibility of the exported attribute and intent-filter element with the Service you are trying to create.

Table 4.4.3: Combination of exported attribute and intent-filter settings (for Service)

Value of exported attribute	Intent-filter defined	Intent-filter not defined
true	(Prohibited)	Public, Partner-only, In-house only
false	(Prohibited)	Private
Not specified	(Prohibited)	(Prohibited)

If the exported attribute is not unspecified in a Service, the question of whether or not the Service is public is determined by whether or not intent filters are defined[15]; however, in this guidebook it is forbidden to set a Service's exported attribute to "unspecified". In general, as mentioned previously, it is best to avoid implementations that rely on the default behavior of any given API; moreover, in cases where explicit methods exist for configuring important security-related settings such as the exported attribute, it is always a good idea to make use of those methods.

In "Table 4.4.3 *Combination of exported attribute and intent-filter settings (for Service)*", all "Intent Filter defined" are set to "(Do not Use)". This is because when a Service is started using an implicit Intent, it is not possible to know which Service responds to the Intent, and a malicious Service may respond.

And the reason why "a defined intent filter and an exported attribute of false" should not be used is that there is a loophole in Android's behavior, and because of how Intent filters work, other application's Services can be called unexpectedly.

Concretely, Android behaves as per below, so it's necessary to consider carefully when application designing.

- When multiple Services define the same content of intent-filter, the definition of Service within application installed earlier is prioritized.
- In case explicit Intent is used, prioritized Service is automatically selected and called by OS.

The system that unexpected call is occurred due to Android's behavior is described in the three figures below. Fig. 4.4.4 is an example of normal behavior that Private Service (application A) can be called by implicit Intent only from the same application. Because only application A defines Intent-filter (action="X" in the Figure), it behaves normally. This is the normal behavior.

[15] If any intent filters are defined then the Service is public; otherwise it is private. For more information, see https://developer.android.com/guide/topics/manifest/service-element#exported

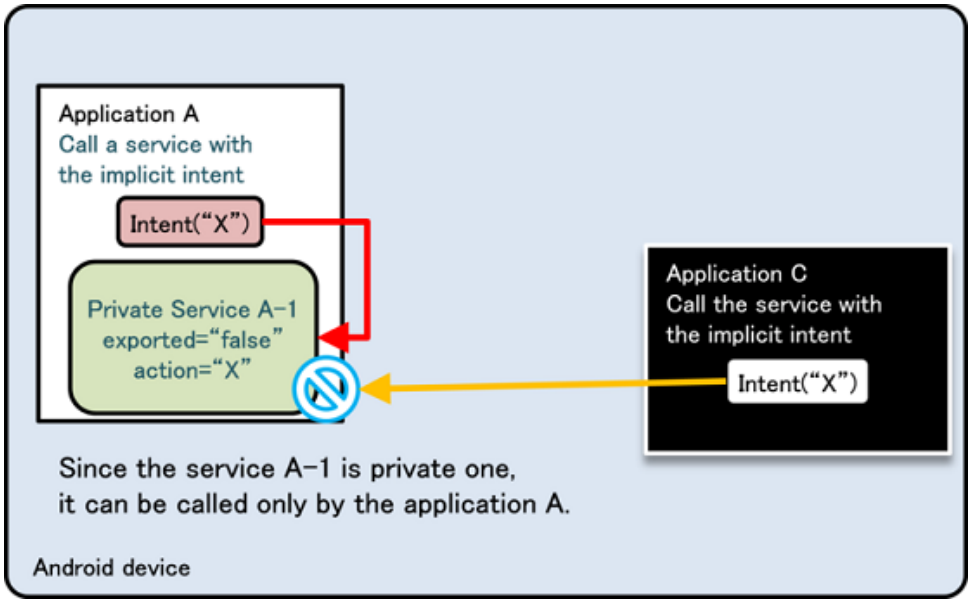


Fig. 4.4.4: An Example of Normal Behavior

Fig. 4.4.5 and Fig. 4.4.6 below show a scenario in which the same Intent filter (action="X") is defined in Application B as well as Application A.

Fig. 4.4.5 shows the scenario that applications are installed in the order, application A -> application B. In this case, when application C sends implicit Intent, calling Private Service (A-1) fails. On the other hand, since application A can successfully call Private Service within the application by implicit Intent as expected, there won't be any problems in terms of security (counter-measure for Malware).

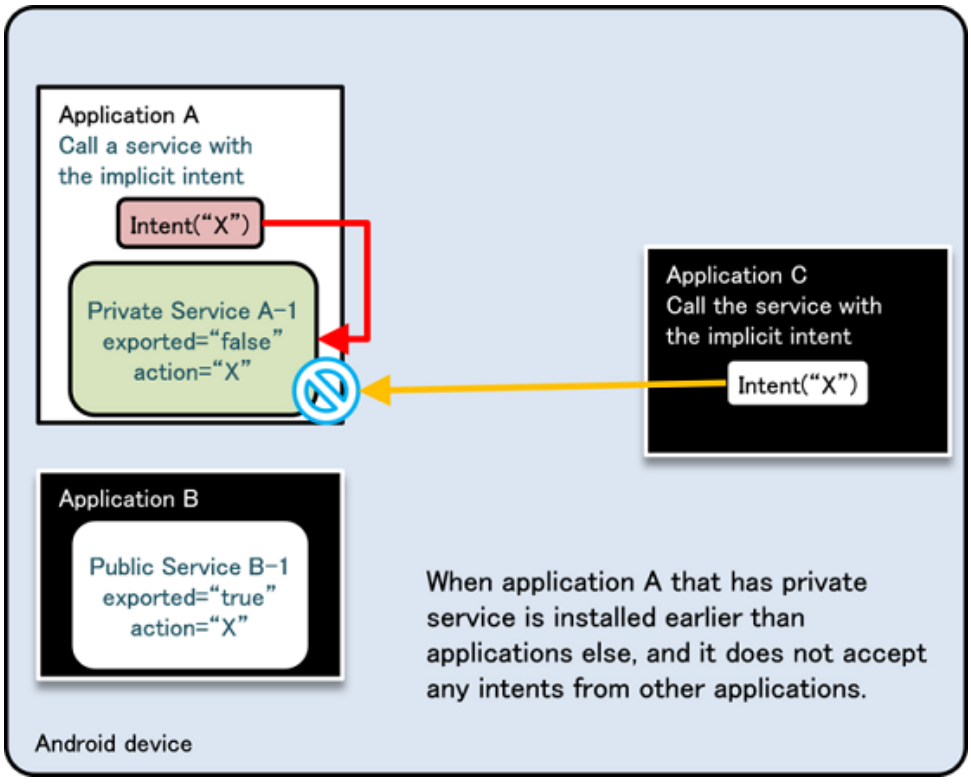


Fig. 4.4.5: Applications are installed in the order, application A -> application B

Fig. 4.4.6 shows the scenario that applications are installed in the order, applicationB -> applicationA. There is a

problem here, in terms of security. It shows an example that applicationA tries to call Private Service within the application by sending implicit Intent, but actually Public Activity (B-1) in application B which was installed earlier, is called. Due to this loophole, it is possible that sensitive information can be sent from applicationA to applicationB. If applicationB is Malware, it will lead the leakage of sensitive information.

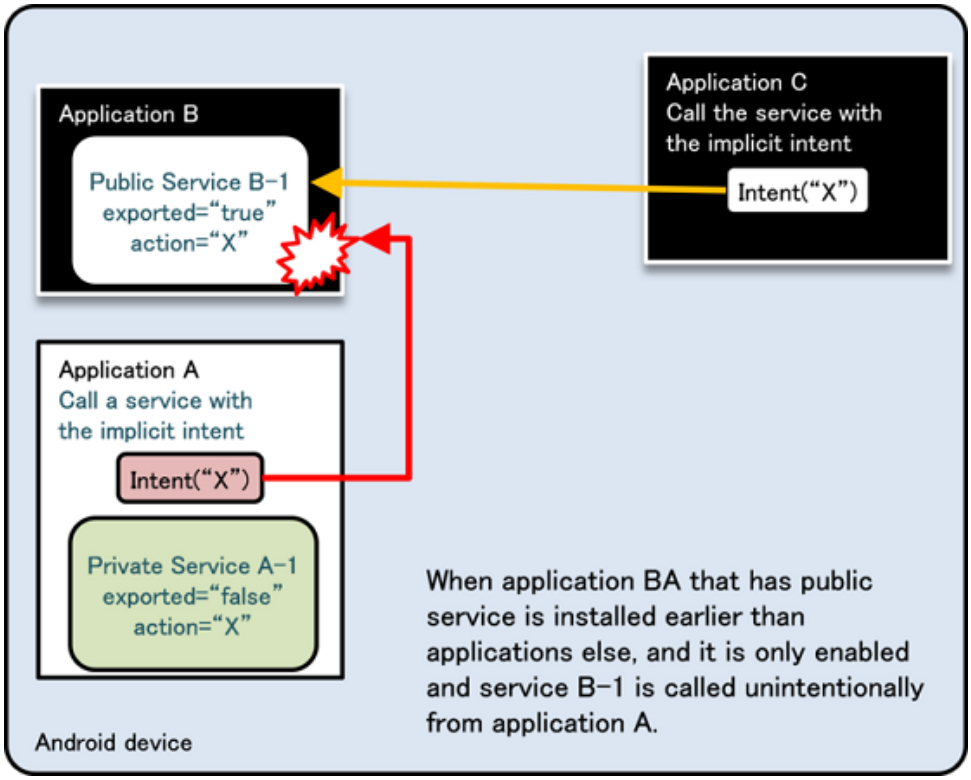


Fig. 4.4.6: Applications are installed in the order, applicationB -> applicationA

As shown above, using Intent filters to send implicit Intents to Private Service may result in unexpected behavior so it is best to avoid this setting.

Note that from Android 12 onwards, the exported attribute is required. For details, please refer to 4.7.3.1. *Component Export Control and Intent Sending Restrictions*.

4.4.3.2 How to Implement Service

Because methods for Service implementation are various and should be selected with consideration of security type which is categorized by sample code, each characteristics are briefly explained. It's divided roughly into the case using startService and the case using bindService. And it's also possible to create Service which can be used in both startService and bindService. Following items should be investigated to determine the implementation method of Service.

- Whether to disclose Service to other applications or not (Disclosure of Service)
- Whether to exchange data during running or not (Mutual sending/receiving data)
- Whether to control Service or not (Launch or complete)
- Whether to execute as another process (communication between processes)
- Whether to execute multiple processes in parallel (Parallel process)

Table 4.4.4 shows category of implementation methods and feasibility of each item.

"NG" stands for impossible case or case that another frame work which is different from the provided function is required.

Table 4.4.4: Category of implementation methods for Service

Category	Disclosure of Service	Mutual sending/receiving data	Control Service (Boot/Exit)	Communication between processes	Parallel process
startService type	OK	NG	OK	OK	NG
IntentService type	OK	NG	NG	OK	NG
local bind type	NG	OK	OK	NG	NG
Messenger bind type	OK	OK	OK	OK	NG
AIDL bind type	OK	OK	OK	OK	OK

startService type

This is the most basic Service. This inherits Service class, and executes processes by onStartCommand.

In user side, specify Service by Intent, and call by startService. Because data such as results cannot be returned to source of Intent directly, it should be achieved in combination with another method such as Broadcast. Please refer to "4.4.1.1. *Creating/Using Private Services*" for the concrete example.

Checking in terms of security should be done by onStartCommand, but it cannot be used for partner only Service since the package name of the source cannot be obtained.

IntentService type

IntentService is the class which was created by inheriting Service. Calling method is same as startService type. Following are characteristics compared with standard service (startService type.)

- Processing Intent is done by onHandleIntent (onStartCommand is not used.)
- It's executed by another thread.
- Process is to be queued.

Call is immediately returned because process is executed by another thread, and process towards Intents is sequentially executed by Queuing system. Each Intent is not processed in parallel, but it is also selectable depending on the product's requirement, as an option to simplify implementation. Since data such as results cannot be returned to source of Intent, it should be achieved in combination with another method such as Broadcast. Please refer to "4.4.1.2. *Creating/Using Public Services*" for the concrete example of implementation.

Checking in terms of security should be done by onHandleIntent, but it cannot be used for partner only Service since the package name of the source cannot be obtained.

local bind type

This is a method to implement local Service which works only within the process same as an application. Define the class which was derived from Binder class, and prepare to provide the feature (method) which was implemented in Service to caller side.

From user side, specify Service by Intent and call Service by using bindService. This is the most simple implementation method among all methods of binding Service, but it has limited usages since it cannot be launched by another process and also Service cannot be disclosed. See project "Service PrivateServiceLocalBind" which is included in Sample code, for the concrete implementation example.

From the security point of view, only private Service can be implemented.

Messenger bind type

This is the method to achieve the linking with Service by using Messenger system.

Since Messenger can be given as a Message destination from Service user side, the mutual data exchanging can be achieved comparatively easily. In addition, since processes are to be queued, it has a characteristic that behaves "thread-safely". Parallel process for each process is not possible, but it is also selectable as an option to simplify the implementation depending on the product's requirement. Regarding user side, specify Service by Intent, and

call Service by using bindService. See "4.4.1.4. *Creating/Using In-house Services*" for the concrete implementation example.

Security check in onBind or by Message Handler is necessary, however, it cannot be used for partner only Service since package name of source cannot be obtained.

AIDL bind type

This is a method to achieve linking with Service by using AIDL system. Define interface by AIDL, and provide features that Service has as a method. In addition, call back can be also achieved by implementing interface defined by AIDL in user side, Multi-thread calling is possible, but it's necessary to implement explicitly in Service side for exclusive process.

User side can call Service, by specifying Intent and using bindService. Please refer to "4.4.1.3. *Creating/Using Partner Services*" for the concrete implementation example.

Security must be checked in onBind for In-house only Service and by each method of interface defined by AIDL for partner only Service.

This can be used for all security types of Service which are described in this Guidebook.

4.4.3.3 Requirement for Specifying of Service Types

When browsing the web in one window and playing music in another, for instance, the more applications running at the same time, the greater the load on the system. Therefore, since Android 8, the creation of a background service from a background application is no longer allowed, and a Context.startForegroundService () method has been added to create the service as a user-recognizable foreground service.

When a service is created by startForegroundService, it is necessary to notify the user that the service is running in the notification drawer while the service in question is operating.

The following is an excerpt of code that uses MediaPlayer to play music in a foreground service.

AndroidManifest.xml

```
<uses-permission android:name="android.permission.FOREGROUND_SERVICE" />
<uses-permission android:name="android.permission.POST_NOTIFICATIONS"/>
```

MainActivity.java

```
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    // ...
    buttonStart.setOnClickListener( v -> {
        Intent intent = new Intent(getApplication(), MediaPlaybackService.class);
        intent.putExtra("REQUEST_CODE", 1);

        // The launcher with the Intent you want to start
        requestPermissionLauncher.launch(Manifest.permission.POST_NOTIFICATIONS);

        // Start Service
        startForegroundService(intent);
    });
```

This code works without any issues up to Android 13, but starting with Android 14, it is necessary to explicitly specify the foreground service type.

The service type is specified in the <service> element of AndroidManifest.xml. An example of the modification is shown below.

AndroidManifest.xml

```
<uses-permission android:name="android.permission.FOREGROUND_SERVICE" />
<uses-permission android:name="android.permission.POST_NOTIFICATIONS"/>
<uses-permission android:name="android.permission.FOREGROUND_SERVICE_MEDIA_PLAYBACK
↪" />


    <!--Omitted-->

    <service
        android:name=".MediaPlaybackService"
        android:foregroundServiceType="mediaPlayback"
        android:exported="false">
    </service>
```

If the service type is not specified in an application targeting Android 14, the exception MissingForegroundService-TypeException will occur when startForegroundService is executed.

```
android.app.MissingForegroundServiceTypeException: Starting FGS without a type ␣
↪callerApp=ProcessRecord{5efdc61 5490:com.example.myapplication/u0a261}␣
↪targetSDK=34
```

The following is a list of service types that are required to be specified.

- camera
- connectedDevice
- dataSync
- health
- location
- mediaPlayback
- mediaProjection
- microphone
- phoneCall
- remoteMessaging
- shortService
- specialUse
- systemExempted

4.4.3.4 Foreground Service Changes

Android 15 introduces several behavioral changes to foreground services with the goal of improving security and privacy. These changes aim to prevent applications from inappropriately using system resources and reduce unintended user behavior. Below are the main changes and the associated security measures:

Data Sync Foreground Service Timeout Behavior

Android 15 introduced a new timeout behavior for the dataSync foreground service. As a result, the dataSync service is allowed to run for a total of 6 hours within a 24-hour period, and the Service.onTimeout(int, int) method is called when that is exceeded.

This change is intended to prevent long-running foreground services from continuing to occupy system resources, thereby adversely affecting the performance of other applications and the system as a whole. It is also intended to reduce the risk of privacy violations due to prolonged unintended user manipulation of data.

Countermeasures

1. Implementing timeout method:

```
override fun onTimeout(startId: Int, timeout: Int) {
    stopSelf(startId)
}
```

2. Service runtime limit: Design the app's dataSync service so that it does not run for more than a total of 6 hours in a 24-hour period. Ensure that the timer is reset only when the user operates the app in the foreground.
3. use of alternative APIs: improve the efficiency and security of background processing by using alternative APIs such as WorkManager instead of the dataSync foreground service.

New Media Processing Foreground Service Type

Android 15 introduces the mediaProcessing foreground service type, which is suitable for media file code conversion. This service is also allowed to run for a total of 6 hours within a 24-hour period, and the Service.onTimeout(int, int) method is called if that is exceeded.

This change is to prevent long periods of media processing from using excessive system resources and negatively affecting the performance of other applications and the overall system.

Countermeasures

1. Implementing timeout method:

```
override fun onTimeout(startId: Int, timeout: Int) {
    stopSelf(startId)
}
```

2. Service Runtime Limit: Design the app's mediaProcessing service to run for a total of more than 6 hours within 24 hours. Ensure that the timer is reset only if the user interacts with the app in the foreground.
3. Using Alternate APIs: Using alternative APIs such as WorkManager without using mediaProcessing foreground services, improve background processing efficiency.
4. Splitting Media Processing: Splitting long media processing into smaller tasks so that each task completes within the time limit
5. User Notification: Notify users of progress when long media processing is required, allowing them to pause or resume processing if necessary.

Restrictions on starting foreground services by BOOT_COMPLETED broadcast receivers

Android 15 restricts certain foreground services (e.g. dataSync, camera, mediaPlayback, etc.) from launching with BOOT_COMPLETED broadcast receivers.

Security Risk, this change is intended to prevent services that start automatically at system startup from causing unintended background behavior by the user, leading to privacy violations and excessive resource consumption.

Countermeasures

1. Compliance with boot restrictions: Design to prevent restricted foreground services from starting with the BOOT_COMPLETED receiver
2. Consider alternatives: If necessary, make sure that the service starts with clear user interaction and permissions

Restrictions on starting foreground services when holding the SYSTEM_ALERT_WINDOW privilegeSYSTEM_ALERT_WINDOW

In Android 15, if an app has the SYSTEM_ALERT_WINDOW permission, it must show a TYPE_APPLICATION_OVERLAY window before starting a foreground service.

This change is intended to prevent the risk of unintended user operations or malicious behavior by apps with SYSTEM_ALERT_WINDOW privilege from freely launching foreground services from the background.

Countermeasures

1. Check the display of the overlay window: make sure the TYPE_APPLICATION_OVERLAY window is displayed before the app starts foreground services

```
// Check if the overlay window is currently being displayed
if (isOverlayWindowVisible()) {
    // Start the foreground service
    startForegroundService()
} else {
    // Show overlay window and then start the foreground service
    showOverlayWindow()
    // Optionally, you could delay the start of the service until the overlay is␣
↪visible
}
```

2. Visibility monitoring: Override View.onWindowVisibilityChanged to receive notifications whenever the window visibility changes.

```
override fun onWindowVisibilityChanged(visibility: Int) {
    super.onWindowVisibilityChanged(visibility)
    // Handle visibility changes
}
```

3. Check process order: Check your app's process order and ensure that there is an active overlay window before starting a foreground service from the background.

4.4.3.5 Strengthening Background Audio Restrictions

In Android 17 and later, to prevent unintended background playback by users, restrictions on audio playback, audio focus requests, and volume change APIs have been strengthened. This restriction applies to all apps running on Android 17, regardless of the app's targetSdkVersion.

Calling these audio APIs while the app is in a background state (a state not in a valid lifecycle) fails silently (is blocked) without an exception being thrown. Because it does not crash with an exception, depending on the implementation, caution is required as it easily leads to bugs in the form of "processing passes, but for some reason no sound plays." Also, in the case of the audio focus API, AUDIOFOCUS_REQUEST_FAILED is returned as the result code, and it fails.

To handle audio in the background, a service configuration that satisfies the following conditions is required:

- All apps running on Android 17: There must be an Activity visible on the screen, or a foreground service (FGS) other than SHORT_SERVICE must be running.
- Apps targeting Android 17 or higher: The running foreground service must meet the while-in-use requirement (permitted only while in use). That is, it must be started in response to user operation, or the FGS must be started while the app is displayed on the screen.

However, only when the app holds exact alarm permissions (such as SCHEDULE_EXACT_ALARM) and plays back using the USAGE_ALARM audio stream, the while-in-use requirement is exceptionally exempt.

The impact on calling apps or apps that perform playback only while the screen is displayed is relatively small, but for apps that play music or podcasts in the background for a long time, it is necessary to thoroughly design to start an appropriate foreground service for media playback triggered by explicit user operations. Flows such as suddenly launching an FGS to play sound immediately after device startup are prone to suppression, and should be avoided.

The following is a code example for verifying how background audio playback is blocked by Android 17 restrictions.

Immediately after calling this function triggered by an operation such as a button tap, return to the home screen to transition the app completely to the background. Even after the specified time (10 seconds here) has elapsed, no audio is played, and no exception such as SecurityException occurs. On Logcat, only system logs such as mediaPlayer went away with unhandled events are output, and it can be confirmed that it fails silently.

```
// Process to play audio after a specified time (milliseconds)
fun playAudioWithDelay(context: Context, delayMillis: Long) {
```

(continues on next page)

(continued from previous page)

```
    Handler(Looper.getMainLooper()).postDelayed({
        try {
            // Get the system default notification sound (simply for use as a␣
↪sound source)
            val defaultUri = RingtoneManager.getDefaultUri(RingtoneManager.TYPE_␣
↪NOTIFICATION)

            // Explicitly set AudioAttributes to "media (music) playback"
            // to avoid exemption from restrictions due to alarm usage and test as␣
↪pure media playback
            val mediaPlayer = MediaPlayer().apply {
                setAudioAttributes(
                    android.media.AudioAttributes.Builder()
                        .setContentType(android.media.AudioAttributes.CONTENT_TYPE_␣
↪MUSIC)
                        .setUsage(android.media.AudioAttributes.USAGE_MEDIA)
                        .build()
                )
                setDataSource(context, defaultUri)
                prepare()
                setOnCompletionListener { it.release() }
            }

            // Start playback (fails silently here if in background state on␣
↪Android 17 or later)
            mediaPlayer.start()
            Log.d("AudioTest", "Executed MediaPlayer.start()")
        } catch (e: Exception) {
            Log.e("AudioTest", "Playback error: ${e.message}")
        }
    }, delayMillis)
}
```

4.5 Using SQLite

Herein after, some cautions in terms of security when creating/operating database by using SQLite. Main points are appropriate setting of access right to database file, and counter-measures for SQL injection. Database which permits reading/writing database file from outside directly (sharing among multiple applications) is not supposed here, but suppose the usage in backend of Content Provider and in an application itself. In addition, it is recommended to adopt counter-measures mentioned below in case of handling not so much sensitive information, though handling a certain level of sensitive information is supposed here.

4.5.1 Sample Code

4.5.1.1 Creating/Operating Database

When handling database in Android application, appropriate arrangements of database files and access right setting (Setting for denying other application's access) can be achieved by using SQLiteOpenHelper[16]. Here is an example of easy application that creates database when it's launched, and executes searching/adding/changing/deleting data through UI. Sample code is what counter-measure for SQL injection is done, to avoid from incorrect SQL being executed against the input from outside.

[16] As regarding file storing, the absolute file path can be specified as the 2nd parameter (name) of SQLiteOpenHelper constructor. Therefore, need attention that the stored files can be read and written by the other applications if the SD Card path is specified.

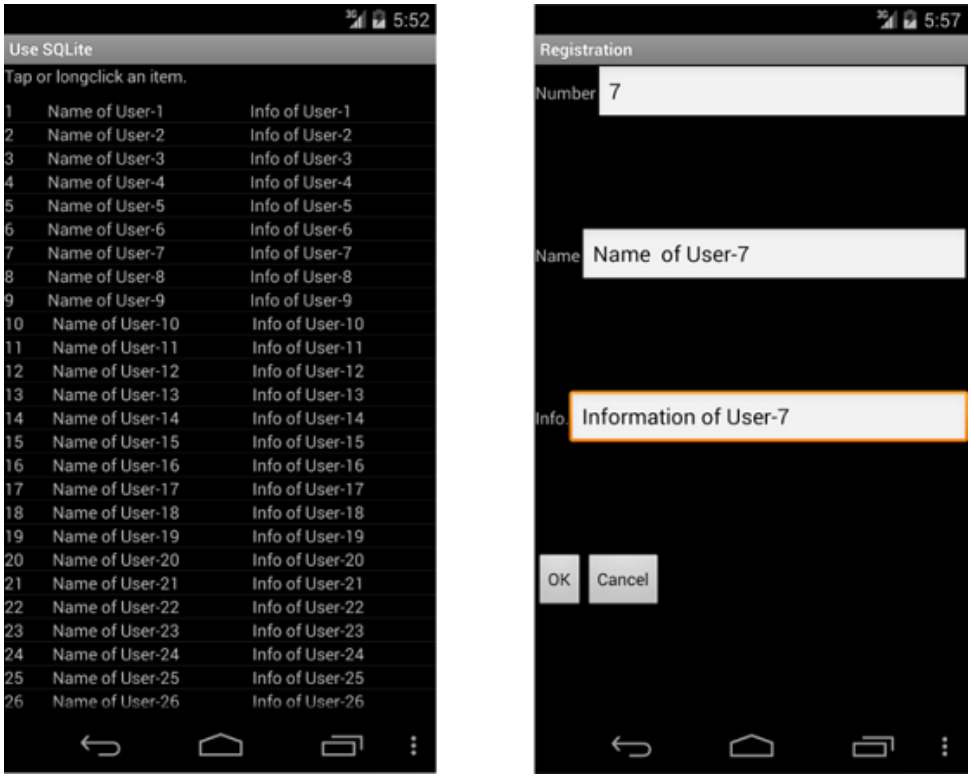


Fig. 4.5.1: Using Database in Android Application

Points:

1. SQLiteOpenHelper should be used for database creation.
2. Use place holder.
3. Validate the input value according the application requirements.

SampleDbOpenHelper.java

```
/*
 * Copyright (C) 2012-2026 Japan Smartphone Security Association
 *
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 *
 *      http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */

package org.jssec.android.sqlite;

import android.content.Context;
import android.database.SQLException;
import android.database.sqlite.SQLiteDatabase;
import android.database.sqlite.SQLiteOpenHelper;
```

(continues on next page)

(continued from previous page)

```
import android.util.Log;
import android.widget.Toast;

public class SampleDbOpenHelper extends SQLiteOpenHelper {
    private SQLiteDatabase mSampleDb;  //Database to store the data to be handled

    public static SampleDbOpenHelper newHelper(Context context)
    {
        //*** POINT 1 *** SQLiteOpenHelper should be used for database creation.
        return new SampleDbOpenHelper(context);
    }

    public SQLiteDatabase getDb() {
        return mSampleDb;
    }

    //Open DB by Writable mode
    public void openDatabaseWithHelper() {
        try {
            if (mSampleDb != null && mSampleDb.isOpen()) {
                if (!mSampleDb.isReadOnly())//  Already opened by writable mode
                    return;
                mSampleDb.close();
            }
            mSampleDb  = getWritableDatabase(); //It's opened here.
        } catch (SQLException e) {
            //In case fail to construct database, output to log
            Log.e(mContext.getClass().toString(),
                    mContext.getString(R.string.DATABASE_OPEN_ERROR_MESSAGE));
            Toast.makeText(mContext,
                           R.string.DATABASE_OPEN_ERROR_MESSAGE,
                           Toast.LENGTH_LONG).show();
        }
    }

    //Open DB by ReadOnly mode.
    public void openDatabaseReadOnly() {
        try {
            if (mSampleDb != null && mSampleDb.isOpen()) {
                if (mSampleDb.isReadOnly())// Already opened by ReadOnly.
                    return;
                mSampleDb.close();
            }
            SQLiteDatabase.openDatabase(mContext.getDatabasePath(CommonData.DBFILE_
↪NAME).getPath(),null, SQLiteDatabase.OPEN_READONLY);
        } catch (SQLException e) {
            //In case failed to construct database, output to log
            Log.e(mContext.getClass().toString(),
                  mContext.getString(R.string.DATABASE_OPEN_ERROR_MESSAGE));
            Toast.makeText(mContext,
                           R.string.DATABASE_OPEN_ERROR_MESSAGE,
                           Toast.LENGTH_LONG).show();
        }
    }

    //Database Close
```

(continues on next page)

(continued from previous page)

```
    public void closeDatabase() {
        try {
            if (mSampleDb != null && mSampleDb.isOpen()) {
                mSampleDb.close();
            }
        } catch (SQLException e) {
            //In case failed to construct database, output to log
            Log.e(mContext.getClass().toString(),
                  mContext.getString(R.string.DATABASE_CLOSE_ERROR_MESSAGE));
            Toast.makeText(mContext,
                           R.string.DATABASE_CLOSE_ERROR_MESSAGE,
                           Toast.LENGTH_LONG).show();
        }
    }

    //Remember Context
    private Context mContext;

    //Table creation command
    private static final String CREATE_TABLE_COMMANDS
        = "CREATE TABLE " + CommonData.TABLE_NAME + " ("
        + "_id INTEGER PRIMARY KEY AUTOINCREMENT, "
        + "idno INTEGER UNIQUE, "
        + "name VARCHAR(" + CommonData.TEXT_DATA_LENGTH_MAX + ") NOT NULL, "
        + "info VARCHAR(" + CommonData.TEXT_DATA_LENGTH_MAX + ")"
        + ");";

    public SampleDbOpenHelper(Context context) {
        super(context, CommonData.DBFILE_NAME, null, CommonData.DB_VERSION);
        mContext = context;
    }

    @Override
    public void onCreate(SQLiteDatabase db) {
        try {
            db.execSQL(CREATE_TABLE_COMMANDS);  //Execute DB construction command
        } catch (SQLException e) {
            //In case failed to construct database, output to log
            Log.e(this.getClass().toString(),
                    mContext.getString(R.string.DATABASE_CREATE_ERROR_MESSAGE));
        }
    }

    @Override
    public void onUpgrade(SQLiteDatabase arg0, int arg1, int arg2) {
        // It's to be executed when database version up. Write processes like data
        // transition.
    }
}
```

DataSearchTask.java(SQLite Database Project)

```
/*
 * Copyright (C) 2012-2026 Japan Smartphone Security Association
 *
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
```

(continues on next page)

(continued from previous page)

```
 * You may obtain a copy of the License at
 *
 *      http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */

package org.jssec.android.sqlite.task;

import org.jssec.android.sqlite.CommonData;
import org.jssec.android.sqlite.DataValidator;
import org.jssec.android.sqlite.MainActivity;
import org.jssec.android.sqlite.R;

import android.database.Cursor;
import android.database.SQLException;
import android.database.sqlite.SQLiteDatabase;
import android.os.AsyncTask;
import android.util.Log;


//Data search task
public class DataSearchTask extends AsyncTask<String, Void, Cursor> {
    private MainActivity    mActivity;
    private SQLiteDatabase  mSampleDB;

    public DataSearchTask(SQLiteDatabase db, MainActivity activity) {
        mSampleDB = db;
        mActivity = activity;
    }

    @Override
    protected Cursor doInBackground(String... params) {
        String  idno = params[0];
        String  name = params[1];
        String  info = params[2];
        String  cols[]  =   {"_id", "idno","name","info"};

        Cursor cur;

        //*** POINT 3 *** Validate the input value according the application
        // requirements.
        if (!DataValidator.validateData(idno, name, info))
            {
                return null;
            }

        //When all parameters are null, execute all search
        if ((idno == null || idno.length() == 0) &&
            (name == null || name.length() == 0) &&
            (info == null || info.length() == 0) ) {
            try {
```

(continues on next page)

(continued from previous page)

```
            cur = mSampleDB.query(CommonData.TABLE_NAME,
                    cols, null, null, null, null, null);
        } catch (SQLException e) {
            Log.e(DataSearchTask.class.toString(),
                    mActivity.getString(R.string.SEARCHING_ERROR_MESSAGE));
            return null;
        }
        return cur;
    }

    //When No is specified, execute searching by No
    if (idno != null && idno.length() > 0) {
        String selectionArgs[] = {idno};

        try {
            //*** POINT 2 *** Use place holder.
            cur = mSampleDB.query(CommonData.TABLE_NAME, cols,
                    "idno = ?", selectionArgs, null, null, null);
        } catch (SQLException e) {
            Log.e(DataSearchTask.class.toString(),
                    mActivity.getString(R.string.SEARCHING_ERROR_MESSAGE));
            return null;
        }
        return cur;
    }

    //When Name is specified, execute perfect match search by Name
    if (name != null && name.length() > 0) {
        String selectionArgs[] = {name};
        try {
            //*** POINT 2 *** Use place holder.
            cur = mSampleDB.query(CommonData.TABLE_NAME, cols,
                    "name = ?", selectionArgs, null, null, null);
        } catch (SQLException e) {
            Log.e(DataSearchTask.class.toString(),
                    mActivity.getString(R.string.SEARCHING_ERROR_MESSAGE));
            return null;
        }
        return cur;
    }

    //Other than above, execute partly match searching with the condition
    // of info.
    //Escape @ in info which was received as input.
    String argString = info.replaceAll("@", "@@");
    //Escape % in info which was received as input.
    argString = argString.replaceAll("%", "@%");
    //Escape _ in info which was received as input.
    argString = argString.replaceAll("_", "@_");
    String selectionArgs[] = {argString};

    try {
        //*** POINT 2 *** Use place holder.
        cur = mSampleDB.query(CommonData.TABLE_NAME, cols,
                "info LIKE '%' || ? || '%' ESCAPE '@'",
                selectionArgs, null, null, null);
```

(continues on next page)

(continued from previous page)

```
        } catch (SQLException e) {
            Log.e(DataSearchTask.class.toString(),
                    mActivity.getString(R.string.SEARCHING_ERROR_MESSAGE));
            return null;
        }
        return cur;
    }

    @Override
    protected void onPostExecute(Cursor resultCur) {
        mActivity.updateCursor(resultCur);
    }
}
```

DataValidator.java

```
/*
 * Copyright (C) 2012-2026 Japan Smartphone Security Association
 *
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 *
 *      http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */

package org.jssec.android.sqlite;

public class DataValidator {
    //Validate the Input value
    //validate numeric characters
    public static boolean validateNo(String idno) {
        //null and blank are OK
        if (idno == null || idno.length() == 0) {
            return true;
        }

        //Validate that it's numeric character.
        try {
            if (!idno.matches("[1-9][0-9]*")) {
                //Error if it's not numeric value
                return false;
            }
        } catch (NullPointerException e) {
            //Detected an error
            return false;
        }

        return true;
    }
```

(continues on next page)

(continued from previous page)

```
    // Validate the length of a character string
    public static boolean validateLength(String str, int max_length) {
        //null and blank are OK
        if (str == null || str.length() == 0) {
            return true;
        }

        //Validate the length of a character string is less than MAX
        try {
            if (str.length() > max_length) {
                //When it's longer than MAX, error
                return false;
            }
        } catch (NullPointerException e) {
            //Bug
            return false;
        }

        return true;
    }

    // Validate the Input value
    public static boolean validateData(String idno, String name, String info) {
        if (!validateNo(idno)) {
            return false;
        }
        if (!validateLength(name, CommonData.TEXT_DATA_LENGTH_MAX)) {
            return false;
        }else if(!validateLength(info, CommonData.TEXT_DATA_LENGTH_MAX)) {
            return false;
        }
        return true;
    }
}
```

4.5.2 Rule Book

Using SQLite, follow the rules below accordingly.

1. *Set DB File Location and Access Right Correctly (Required)*
2. *Use Content Provider for Access Control When Sharing DB Data with Other Application (Required)*
3. *Place Holder Must Be Used in the Case Handling Variable Parameter during DB Operation. (Required)*

4.5.2.1 Set DB File Location and Access Right Correctly (Required)

Considering the protection of DB file data, DB file location and access right setting is the very important elements that need to be considered together.

For example, even if file access right is set correctly, a DB file can be accessed from anybody in case that it is arranged in a location which access right cannot be set, e.g. SD card. And in case that it's arranged in application directory, if the access right is not correctly set, it will eventually allow the unexpected access. Following are some points to be met regarding the correct allocation and access right setting, and the methods to realize them.

About location and access right setting, considering in terms of protecting DB file (data), it's necessary to execute 2 points as per below.

1. Location

Locate in file path that can be obtained by Context#getDatabasePath(String name), or in some cases, directory that can be obtained by Context#getFilesDir[17].

2. Access right

Set to MODE_PRIVATE (= it can be accessed only by the application which creates file) mode.

By executing following 2 points, DB file which cannot be accessed by other applications can be created. Here are some methods to execute them.

1. Use SQLiteOpenHelper

2. Use Context#openOrCreateDatabase

When creating DB file, SQLiteDatabase#openOrCreateDatabase can be used. However, when using this method, DB files which can be read out from other applications are created, in some Android smartphone devices. So it is recommended to avoid this method, and using other methods. Each characteristics for the above 2 methods are as per below.

Using SQLiteOpenHelper

When using SQLiteOpenHelper, developers don't need to be worried about many things. Create a class derived from SQLiteOpenHelper, and specify DB name (which is used for file name)[18] to constructer's parameter, then DB file which meets above security requirements, are to be created automatically.

Refer to specific usage method for "4.5.1.1. *Creating/Operating Database*" for how to use.

Using Context#openOrCreateDatabase

When creating DB by using Context#openOrCreateDatabase method, file access right should be specified by option, in this case specify MODE_PRIVATE explicitly.

Regarding file arrangement, specifying DB name (which is to be used to file name) can be done as same as SQLiteOpenHelper, a file is to be created automatically, in the file path which meets the above mentioned security requirements. However, full path can be also specified, so it's necessary to pay attention that when specifying SD card, even though specifying MODE_PRIVATE, other applications can also access.

Example to execute access permission setting to DB explicitly: MainActivity.java

```
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);

    //Construct database
    try {
        //Create DB by setting MODE_PRIVATE
        db = Context.openOrCreateDatabase("Sample.db", MODE_PRIVATE, null);
    } catch (SQLException e) {
        //In case failed to construct DB, log output
        Log.e(this.getClass().toString(),
              getString(R.string.DATABASE_OPEN_ERROR_MESSAGE));
        return;
    }
    //Omit other initial process
}
```

There are three possible settings for access privileges: MODE_PRIVATE, MODE_WORLD_READABLE, and MODE_WORLD_WRITEABLE. These constants can be specified together by "OR" operator. However, all settings other than MODE_PRIVATE are deprecated in API Level 17 and later versions, and will result in a security exception

[17] Both methods provide the path under (package) directory which is able to be read and written only by the specified application.

[18] (Undocumented in Android reference) Since the full file path can be specified as the database name in SQLiteOpenHelper implementation, need attention that specifying the place (path) which does not have access control feature (e.g. SD cards) unintentionally.

in API Level 24 and later versions. Even for apps intended for API Level 15 and earlier, it is generally best not to use these flags[19].

- MODE_PRIVATE Only creator application can read and write
- MODE_WORLD_READABLE Creator application can read and write, Others can only read in
- MODE_WORLD_WRITEABLE Creator application can read and write, Others can only write in

4.5.2.2 Use Content Provider for Access Control When Sharing DB Data with Other Application (Required)

The method to share DB data with other application is that create DB file as WORLD_READABLE, WORLD_WRITEABLE, to other applications to access directly. However, this method cannot limit applications which access to DB or operations to DB, so data can be read-in or written by unexpected party (application). As a result, it can be considered that some problems may occur in confidentiality or consistency of data, or it may be an attack target of Malware.

As mentioned above, when sharing DB data with other applications in Android, it's strongly recommended to use Content Provider. By using Content Provider, there are some merits, not only the merits from the security point of view which is the access control on DB can be achieved, but also merits from the designing point of view which is DB scheme structure can be hidden into Content Provider.

4.5.2.3 Place Holder Must Be Used in the Case Handling Variable Parameter during DB Operation. (Required)

In the sense that preventing from SQL injection, when incorporating the arbitrary input value to SQL statement, placeholder should be used. There are 2 methods as per below to execute SQL using placeholder.

1. Get SQLiteStatement by using SQLiteDatabase#compileStatement(), and after that place parameter to placeholder by using SQLiteStatement#bindString() or bindLong() etc.
2. When calling execSQL(), insert(), update(), delete(), query(), rawQuery() and replace() in SQLiteDatabase class, use SQL statement which has placeholder.

In addition, when executing SELECT command, by using SQLiteDatabase#compileStatement(), there is a limitation that "only the top 1 element can be obtained as a result of SELECT command", so usages are limited.

In either method, the data content which is given to placeholder is better to be checked in advance according the application requirements. Following is the further explanation for each method.

When Using SQLiteDatabase#compileStatement():

Data is given to placeholder in the following steps.

1. Get the SQL statement which includes placeholder by using SQLiteDatabase#compileStatement(), as SQLiteStatement.
2. Set the created as SQLiteStatement objects to placeholder by using the method like bindLong() and bindString().
3. Execute SQL by method like execute() of ExecSQLiteStatement object.

Use case of placeholder: DataInsertTask.java (an extra)

```
//Adding data task
public class DataInsertTask extends AsyncTask<String, Void, Void> {
    private MainActivity    mActivity;
    private SQLiteDatabase  mSampleDB;

    public DataInsertTask(SQLiteDatabase db, MainActivity activity) {
```

(continues on next page)

[19] For more information as to MODE_WORLD_READABLE and MODE_WORLD_WRITEABLE and points of caution regarding their use, see Section "4.6.3.2. *Access Permission Setting for the Directory*".

(continued from previous page)

```
        mSampleDB = db;
        mActivity = activity;
    }

    @Override
    protected Void doInBackground(String... params) {
        String  idno = params[0];
        String  name = params[1];
        String  info = params[2];

        // *** POINT 3 *** Validate the input value according the application
        // requirements.
        if (!DataValidator.validateData(idno, name, info))
        {
            return null;
        }
        // *** POINT 2 *** Use place holder
        // Adding data task
        String commandString =
           "INSERT INTO " + CommonData.TABLE_NAME + " (idno, name, info) VALUES (?,
↪ ?, ?)";
        SQLiteStatement sqlStmt = mSampleDB.compileStatement(commandString);
        sqlStmt.bindString(1, idno);
        sqlStmt.bindString(2, name);
        sqlStmt.bindString(3, info);
        try {
            sqlStmt.executeInsert();
        } catch (SQLException e) {
            Log.e(DataInsertTask.class.toString(),
                  mActivity.getString(R.string.UPDATING_ERROR_MESSAGE));
        } finally {
            sqlStmt.close();
        }
        return null;
    }
    ... Abbreviation ...
}
```

This is a type that SQL statement to be executed as object is created in advance, and parameters are allocated to it. The process to execute is fixed, so there's no room for SQL injection to occur. In addition, there is a merit that process efficiency is enhanced by reutilizing SQLiteStatement object.

In the Case Using Method for Each Process which SQLiteDatabase provides:

There are 2 types of DB operation methods that SQLiteDatabase provides. One is what SQL statement is used, and another is what SQL statement is not used. Methods that SQL statement is used are SQLiteDatabase#execSQL()/rawQuery() and it's executed in the following steps.

1. Prepare SQL statement which includes placeholder.
2. Create data to allocate to placeholder.
3. Send SQL statement and data as parameter, and execute a method for process.

On the other hand, SQLiteDatabase#insert()/update()/delete()/query()/replace() is the method that SQL statement is not used. When using them, data should be sent as per the following steps.

1. In case there's data to insert/update to DB, register to ContentValues.
2. Send ContentValues as parameter, and execute a method for each process (In the following example, SQLiteDatabase#insert())

Use case of metod for each process (SQLiteDatabase#insert())

```
private SQLiteDatabase  mSampleDB;
private void addUserData(String idno, String name, String info) {

    // Validity check of the value(Type, range), escape process
    if (!validateInsertData(idno, name, info)) {
        // If failed to pass the validation, log output
        Log.e(this.getClass().toString(),
              getString(R.string.VALIDATION_ERROR_MESSAGE));
        return;
    }

    // Prepare data to insert
    ContentValues insertValues = new ContentValues();
    insertValues.put("idno", idno);
    insertValues.put("name", name);
    insertValues.put("info", info);

    // Execute Insert
    try {
        mSampleDb.insert("SampleTable", null, insertValues);
    } catch (SQLException e) {
        Log.e(this.getClass().toString(),
              getString(R.string.DB_INSERT_ERROR_MESSAGE));
        return;
    }
}
```

In this example, SQL command is not directly written, for instead, a method for inserting which SQLiteDatabase provides, is used. SQL command is not directly used, so there's no room for SQL injection in this method, too.

4.5.3 Advanced Topics

4.5.3.1 When Using Wild Card in LIKE Predicate of SQL Statement, Escape Process Should Be Implemented

When using character string which includes wild card (%, _) of LIKE predicate, as input value of place holder, it will work as a wild card unless it is processed properly, so it's necessary to implement escape process in advance according the necessity. It is the case which escape process is necessary that wild card should be used as a single character ("%" or "_").

The actual escape process is executed by using ESCAPE clause as per below sample code.

Example of ESCAPE process in case of using LIKE

```
// Data search task
public class DataSearchTask extends AsyncTask<String, Void, Cursor> {
    private MainActivity        mActivity;
    private SQLiteDatabase      mSampleDB;
    private ProgressDialog      mProgressDialog;

    public DataSearchTask(SQLiteDatabase db, MainActivity activity) {
        mSampleDB = db;
        mActivity = activity;
    }

    @Override
```

(continues on next page)

(continued from previous page)

```
    protected Cursor doInBackground(String... params) {
        String  idno = params[0];
        String  name = params[1];
        String  info = params[2];
        String  cols[]  =   {"_id", "idno","name","info"};

        Cursor cur;

        ... Abbreviation ...

        // Execute like search(partly match) with the condition of info
        // Point: Escape process should be performed on characters
        // which is applied to wild card
        // Escape @ in info which was received as input
        String argString = info.replaceAll("@", "@@");
        // Escape % in info which was received as input
        argString = argString.replaceAll("%", "@%");
        // Escape _ in info which was received as input
        argString = argString.replaceAll("_", "@_");
        String selectionArgs[] = {argString};

        try {
            // Point: Use place holder
            cur = mSampleDB.query("SampleTable", cols,
                                  "info LIKE '%' || ? || '%' ESCAPE '@'",
                                  selectionArgs, null, null, null);
        } catch (SQLException e) {
            Toast.makeText(mActivity,
                R.string.SERCHING_ERROR_MESSAGE, Toast.LENGTH_LONG).show();
            return null;
        }
        return cur;
    }

    @Override
    protected void onPostExecute(Cursor resultCur) {
        mProgressDialog.dismiss();
        mActivity.updateCursor(resultCur);
    }
}
```

4.5.3.2 Use External Input to SQL Command in which Place Holder Cannot Be Used

When executing SQL statement which process targets are DB objects like table creation/deletion etc., placeholder cannot be used for the value of table name. Basically, DB should not be designed using arbitrary character string which was input from outside in case that placeholder cannot be used for the value.

When placeholder cannot be used due to the restriction of specifications or features, whether the Input value is dangerous or not, should be verified before execution, and it's necessary to implement necessary processes.

Basically,

1. When using as character string parameter, escape or quote process for character should be made.
2. When using as numeric value parameter, verify that characters other than numeric value are not included.
3. When using as identifier or command, verify whether characters which cannot be used are not included, along with 1.

should be executed.

> Reference: https://www.ipa.go.jp/security/vuln/documents/website_security_sql.pdf (Japanese)

4.5.3.3 Take a Countermeasure that Database Is Not Overwritten Unexpectedly

In case getting instance of DB by SQLiteOpenHelper#getReadableDatabase, getWriteableDatabase, DB is to be opened in readable/WRITEABLE state by using either method[20]. In addition, it's same to Context#openOrCreateDatabase, SQLiteDatabase#openOrCreateDatabase, etc.

It means that contents of DB may be overwritten unexpectedly by application operation or by defects in implementation. Basically, it can be supported by the application's spec and range of implementation, but when implementing the function which requires only read in function like application's searching function etc., opening database by read-only, it may lead to simplify designing or inspection and furthermore, lead to enhance application quality, so it's recommended depends on the situation.

Specifically, open database by specifying OPEN_READONLY to SQLiteDatabase#openDatabase.

Open database by read-only.

```
... Abbreviation ...

// Open DB(DB should be created in advance)
SQLiteDatabase db
    = SQLiteDatabase.openDatabase(SQLiteDatabase.getDatabasePath("Sample.db"),
                                  null, OPEN_READONLY);
```

> Reference: [https://developer.android.com/reference/android/database/sqlite/SQLiteOpenHelper.html - getReadableDatabase()](https://developer.android.com/reference/android/database/sqlite/SQLiteOpenHelper.html#getReadableDatabase())

4.5.3.4 Verify the Validity of Input/Output Data of DB, According to Application's Requirement

SQLite is the database which is tolerant types, and it can store character type data into columns which is declared as Integer in DB. Regarding data in database, all data including numeric value type is stored in DB as character data of plain text. So searching of character string type, can be executed to Integer type column. (LIKE '%123%' etc.) In addition, the limitation for the value in SQLite (validity verification) is untrustful since data which is longer than limitation can be input in some case, e.g. VARCHAR(100).

So, applications which use SQLite, need to be very careful about this characteristics of DB, and it is necessary take actions according to application requirements, not to store unexpected data to DB or not to get unexpected data. Countermeasures are as per below 2 points.

1. When storing data in database, verify that type and length are matched.
2. When getting the value from database, verify whether data is beyond the supposed type and length, or not.

Following is an example of the code which verifies that the Input value is more than 1.

Verify that the Input value is more than 1 (Extract from MainActivity.java)

```
public class MainActivity extends Activity {

    ... Abbreviation ...

    // Process for adding
    private void addUserData(String idno, String name, String info) {
```

(continues on next page)

[20] getReableDatabase() returns the same object which can be got by getWritableDatabase. This spec is, in case writable object cannot be generated due to disc full etc., it will return Read- only object. (getWritableDatabase() will be execution error under the situation like disc full etc.)

(continued from previous page)

```
        // Check for No
        if (!validateNo(idno, CommonData.REQUEST_NEW)) {
            return;
        }

        // Inserting data process
        DataInsertTask task = new DataInsertTask(mSampleDb, this);
        task.execute(idno, name, info);
    }

    ... Abbreviation ...

    private boolean validateNo(String idno, int request) {
        if (idno == null || idno.length() == 0) {
            if (request == CommonData.REQUEST_SEARCH) {
                // When search process, unspecified is considered as OK.
                return true;
            } else {
                // Other than search process, null and blank are error.
                Toast.makeText(this,
                    R.string.IDNO_EMPTY_MESSAGE, Toast.LENGTH_LONG).show();
                return false;
            }
        }

        // Verify that it's numeric character
        try {
            // Value which is more than 1
            if (!idno.matches("[1-9][0-9]*")) {
                // In case of not numeric character, error
                Toast.makeText(this, R.string.IDNO_NOT_NUMERIC_MESSAGE,
                    Toast.LENGTH_LONG).show();
                return false;
            }
        } catch (NullPointerException e) {
            // It never happen in this case
            return false;
        }
        return true;
    }

    ... Abbreviation ...
}
```

4.5.3.5 Consideration - the Data Stored into Database

In SQLite implementation, when storing data to file is as per below.

- All data including numeric value type are stored into DB file as character data of plain text.
- When executing data deletion to DB, data itself is not deleted form DB file. (Only deletion mark is added.)
- When updating data, data before updating has not been deleted, and still remains there in DB file.

So, the information which "must have" been deleted may still remain in DB file. Even in this case, take countermeasures according this Guidebook, and when Android security function is enabled, data/file may not be directly accessed by the third party including other applications. However, considering the case that files are picked out by

passing through Android's protection system like root privilege is taken, in case the data which gives huge influence on business is stored, data protection which doesn't depend on Android protection system, should be considered.

As above reasons, the important data which is necessary to be protected even when device's root privilege is taken, should not be stored in DB of SQLite, as it is. In case need to store the important data, it's necessary to implement counter-measures, or encrypt overall DB.

When encryption is necessary, there are so many issues that are beyond the range of this Guidebook, like handling the key which is used for encryption or code obfuscation, so as of now it's recommended to consult the specialist when developing an application which handles data that has huge business impact.

Please refer to "4.5.3.6. *[Reference] Encrypt SQLite Database (SQLCipher for Android)*" library which encrypts database is introduced here.

4.5.3.6 [Reference] Encrypt SQLite Database (SQLCipher for Android)

Developed by Zetetic LLC, SQLCipher provides transparent 256-bit AES encryption of SQLite databases. It is an SQLite extension library implemented in C language, and it uses OpenSSL for encryption. It also provides APIs for Obj-C, Java, Python, and other languages. In addition to the commercial version, an open source version (called “community edition”) is also available, and it can be used for commercial purposes with a BSD license. It supports a wide range of platforms including Windows, Linux, macOS, and more, and in the mobile space, besides Android, it is also widely used in Nokia / QT and Apple’s iOS.

Among these versions, SQLCipher for Android was packaged specifically for Android use[21]. Although content can be created by compiling from the available source code, a library is also distributed in AAR format (android-database-sqlcipher-xxxx.aar), and this may convenient for simple usage[22]. Some standard SQLite APIs can be changed to match SQLCipher to enable developers to use databases encrypted with the same coding as usual. This section provides a brief introduction of how to use libraries in AAR format.

Reference: https://www.zetetic.net/sqlcipher/

How to Use

The following procedure is used in Android Studio to enable use of SQLCipher.

1. Place android-database-sqlcipher-3.5.9.aar in the libs directory of the application. [(https://www.zetetic.net/sqlcipher/open-source/){]}(https://www.zetetic.net/sqlcipher/open-source/)
2. Specify the dependency in app/gradle.

```
dependencies {
    :
    implementation 'net.zetetic:android-database-sqlcipher:3.5.9@aar'
    :
}
```

3. Instead of the normal android.database.sqlite.*, import net.sqlcipher.database.*. (The android.database.Cursor can be used without any changes.)

4. Before using the database, load and initialize the library, and specify the password when opening the database.

The code shown below is used to execute the initialization process for using the database. Before an activity uses the database, it is assumed that SQLCipherInitializer.Initialize() is called. First, SQLiteDatabase.loadLibs(this) is called, and then the required library is loaded and initialized. Also, when a database is opened using SQLiteDatabase.openOrCreateDatabase(), the password is passed. The database is encrypted using an encryption key generated based on the password provided here. The key point here is that a database created in plain text cannot be converted into an encrypted database later, and the password must be specified when the database is created.

[21] https://github.com/sqlcipher/android-database-sqlcipher

[22] In these explanations, xxxx is the version number of the library, and the latest version at the time of this writing was 3.5.9. The explanations below assume use of this version.

```
package android.jssec.org.samplesqlcipher;
import android.content.Context;
// instead of the normal android.database.sqlite*, import net.sqlcipher.database*
import net.sqlcipher.database.SQLiteDatabase;
import java.io.File;
public class SQLCipherInitializer {
    static SQLiteDatabase Initialize(Context ctx, String dbName, String password) {
        // before using DB, load neccessary libraries and initialize
        SQLiteDatabase.loadLibs(ctx);
        // create databe file uder the package local directory
        File databaseFile = ctx.getDatabasePath(dbName);
        // password must be specified when the databse is created
        return SQLiteDatabase.openOrCreateDatabase(databaseFile, password, null);
    }
}
```

The above shows an example using SQLiteDatabase.openOrCreateDatabase(), but the SQLiteOpenHelper#getWritableDatabase() and SQLiteOpenHelper#getReadableDatabase() APIs have been modified so that the password can be passed as an argument. In either case, if null is specified for the password, a normal SQLite database is created without encrypting the database.

Another key point is that Context#openOrCreateDatabase() cannot be used. As a result, it is not possible to force setting of protection mode for the database files or force creation of a database in the local directory of the package. Consequently, when a database is created using SQLiteDatabase.openOrCreateDatabase(), at the minimum, as shown in the example above, it is recommended that a database be created in the database directory of the package itself using getDatabasePath(). On the other hand, there are no modifications to the constructor API of SQLiteOpenHelper, and if this is used, a database is created in the local directory of the package in the same way as android.database.sqlite.SQLiteOpenHelper.

4.6 Handling Files

According to the Android security design philosophy, files should be used only for the purpose of making information persistent or saving it temporarily (caching), and as a general rule, they should be kept private. When exchanging information between apps, rather than allowing direct access to files, the information within files should be exchanged through inter-app communication mechanisms such as Content Providers and Services. This also enables access control between apps.

Because external storage devices such as SD cards do not allow for sufficient access control, their use should be limited to cases where it is functionally essential, such as when handling large files or moving information to another location (e.g., a PC). Fundamentally, files containing sensitive information must not be placed on external storage devices. If sensitive information must absolutely be stored in a file on an external storage device, countermeasures such as encryption are required, but these are not covered here.

4.6.1 Sample Code

As mentioned above, files should generally be kept private. However, due to various circumstances, there are times when it is necessary to allow other apps to directly read and write files. A classification and comparison of file types from a security perspective is shown in Table 4.6.1. Files are classified into four types based on the combination of the storage location and access permissions for other apps. In the following sections, we will show sample codes and provide explanations for each of these file classifications.

Table 4.6.1: Classification and comparison of files from a security perspective

File classification	Access permissions for other apps	Storage location	Overview
Private files	None	Within the app directory	• Can only be read and written within the app. • Can handle sensitive information. • As a general rule, files should be of this type.
Public read-only files	Read	Within the app directory	• Can be read by other apps and users. • Handles information that can be made public (viewed) outside the app. • The MODE_WORLD_READABLE variable for creating public files was deprecated in API Level 17, and throws a security exception in API Level 24 and higher.
Public read/write files	Read/Write	Within the app directory	• Can be read and written by other apps and users. • Use should be avoided from both security and app design perspectives.
External storage files	Read/Write	External storage devices such as SD cards	• Access rights cannot be controlled. • Files can always be read, written, and deleted by other apps and users. • Use should be kept to the minimum necessary. • Can handle relatively large files. • Use scoped storage from API Level 29 and higher.
External storage files (API Level 29 and higher)	Read/Write	External storage devices such as SD cards	• A scoped view for external storage allows apps to save files specifically for that app in an app-specific directory. • Accessing files created by other apps requires both of the following conditions: - The app is granted the READ_EXTERNAL_STORAGE permission. - The file is stored in MediaStore.Images, MediaStore.Video, or MediaStore.Audio. • To access other files (including those in the Downloads directory), the Storage Access Framework must be used. • Can handle relatively large files.

4.6.1.1 Using Private Files

This applies when handling files that are read and written only within the same app, which is a secure way to use files. Regardless of whether the information stored in the file can be made public, as a general rule, files should be kept private as much as possible, and necessary information exchange with other apps should be performed using other Android mechanisms (Content Providers, Services).

Points: 1. Create the file within the app directory. 2. Set the file's access permissions to private mode so that other apps cannot use it. 3. Sensitive information can be stored. 4. Check the safety of the information stored (or to be stored) in the file, regardless of its source.

PrivateFileActivity.java

```
/*
 * Copyright (C) 2012-2026 Japan Smartphone Security Association
 *
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
```

(continues on next page)

(continued from previous page)

```
 * You may obtain a copy of the License at
 *
 *      http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */

package org.jssec.android.file.privatefile;

import java.io.File;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.IOException;

import android.app.Activity;
import android.os.Bundle;
import android.view.View;
import android.widget.TextView;

public class PrivateFileActivity extends Activity {

    private TextView mFileView;

    private static final String FILE_NAME = "private_file.dat";

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.file);

        mFileView = (TextView) findViewById(R.id.file_view);
    }

    /**
     * Create file process
     *
     * @param view
     */
    public void onCreateFileClick(View view) {
        FileOutputStream fos = null;
        try {
            // *** POINT 1 *** Files must be created in application directory.
            // *** POINT 2 *** The access privilege of file must be set private
            // mode in order not to be used by other applications.
            fos = openFileOutput(FILE_NAME, MODE_PRIVATE);

            // *** POINT 3 *** Sensitive information can be stored.
            // *** POINT 4 *** Regarding the information to be stored in files,
            // handle file data carefully and securely.
            // Omitted, since this is a sample. Please refer to
            // "3.2 Handling Input Data Carefully and Securely."
```

(continues on next page)

(continued from previous page)

```
            fos.write(new String("Not sensotive information (File Activity)\n")
                                .getBytes());
        } catch (FileNotFoundException e) {
            mFileView.setText(R.string.file_view);
        } catch (IOException e) {
            android.util.Log.e("PrivateFileActivity",
                               "failed to read file");
        } finally {
            if (fos != null) {
                try {
                    fos.close();
                } catch (IOException e) {
                    android.util.Log.e("PrivateFileActivity",
                                       "failed to close file");
                }
            }
        }

        finish();
    }

    /**
     * Read file process
     *
     * @param view
     */
    public void onReadFileClick(View view) {
        FileInputStream fis = null;
        try {
            fis = openFileInput(FILE_NAME);

            byte[] data = new byte[(int) fis.getChannel().size()];

            fis.read(data);

            String str = new String(data);

            mFileView.setText(str);
        } catch (FileNotFoundException e) {
            mFileView.setText(R.string.file_view);
        } catch (IOException e) {
            android.util.Log.e("PrivateFileActivity",
                               "failed to read file");
        } finally {
            if (fis != null) {
                try {
                    fis.close();
                } catch (IOException e) {
                    android.util.Log.e("PrivateFileActivity",
                                       "failed to close file");
                }
            }
        }
    }

    /**
```

(continues on next page)

(continued from previous page)

```
     * Delete file process
     *
     * @param view
     */
    public void onDeleteFileClick(View view) {

        File file = new File(this.getFilesDir() + "/" + FILE_NAME);
        file.delete();

        mFileView.setText(R.string.file_view);
    }
}
```

PrivateUserActivity.java

```
/*
 * Copyright (C) 2012-2026 Japan Smartphone Security Association
 *
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 *
 *      http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */

package org.jssec.android.file.privatefile;

import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.IOException;

import android.app.Activity;
import android.content.Intent;
import android.os.Bundle;
import android.view.View;
import android.widget.TextView;

public class PrivateUserActivity extends Activity {

    private TextView mFileView;

    private static final String FILE_NAME = "private_file.dat";

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.user);
        mFileView = (TextView) findViewById(R.id.file_view);
    }
```

(continues on next page)

(continued from previous page)

```
    private void callFileActivity() {
        Intent intent = new Intent();
        intent.setClass(this, PrivateFileActivity.class);

        startActivity(intent);
    }

    /**
     *  Call file Activity process
     *
     * @param view
     */
    public void onCallFileActivityClick(View view) {
        callFileActivity();
    }

    /**
     *  Read file process
     *
     * @param view
     */
    public void onReadFileClick(View view) {
        FileInputStream fis = null;
        try {
            fis = openFileInput(FILE_NAME);

            byte[] data = new byte[(int) fis.getChannel().size()];

            fis.read(data);

            // *** POINT 4 *** Regarding the information to be stored in files,
            // handle file data carefully and securely.
            // Omitted, since this is a sample. Please refer to
            // "3.2 Handling Input Data Carefully and Securely."
            String str = new String(data);

            mFileView.setText(str);
        } catch (FileNotFoundException e) {
            mFileView.setText(R.string.file_view);
        } catch (IOException e) {
            android.util.Log.e("PrivateUserActivity",
                               "failed to read file");
        } finally {
            if (fis != null) {
                try {
                    fis.close();
                } catch (IOException e) {
                    android.util.Log.e("PrivateUserActivity",
                                       "failed to close file");
                }
            }
        }
    }

    /**
     * Rewrite file process
```

(continues on next page)

(continued from previous page)

```
     *
     * @param view
     */
    public void onWriteFileClick(View view) {
        FileOutputStream fos = null;
        try {
            // *** POINT 1 *** Files must be created in application directory.
            // *** POINT 2 *** The access privilege of file must be set private
            // mode in order not to be used by other applications.
            fos = openFileOutput(FILE_NAME, MODE_APPEND);

            // *** POINT 3 *** Sensitive information can be stored.
            // *** POINT 4 *** Regarding the information to be stored in files,
            // handle file data carefully and securely.
            // Omitted, since this is a sample. Please refer to
            // "3.2 Handling Input Data Carefully and Securely."
            fos.write(new String("Sensitive information (User Activity)\n").
↪getBytes());
        } catch (FileNotFoundException e) {
            mFileView.setText(R.string.file_view);
        } catch (IOException e) {
            android.util.Log.e("PrivateUserActivity",
                               "failed to read file");
        } finally {
            if (fos != null) {
                try {
                    fos.close();
                } catch (IOException e) {
                    android.util.Log.e("PrivateUserActivity",
                                       "failed to close file");
                }
            }
        }

        callFileActivity();
    }
}
```

4.6.1.2 Using Public Read Only Files

This is the case to use files to disclose the contents to unspecified large number of applications. If you implement by following the below points, it's also comparatively safe file usage method. Note that using the MODE_WORLD_READABLE variable to create a public file is deprecated in API Level 17 and later versions, and will trigger a security exception in API Level 24 and later versions, therefore, the following sample code does not work ;thus file-sharing methods using Content Provider are preferable.

Points:

1. Files must be created in application directory.
2. The access privilege of file must be set to read only to other applications.
3. Sensitive information must not be stored.
4. Regarding the information to be stored in files, handle file data carefully and securely.

```
PublicFileActivity.java
/*
```

(continues on next page)

(continued from previous page)

```
 * Copyright (C) 2012-2026 Japan Smartphone Security Association
 *
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 *
 *      http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */

package org.jssec.android.file.publicfile.readonly;

import java.io.File;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.IOException;

import android.app.Activity;
import android.os.Bundle;
import android.view.View;
import android.widget.TextView;

public class PublicFileActivity extends Activity {

    private TextView mFileView;

    private static final String FILE_NAME = "public_file.dat";

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.file);

        mFileView = (TextView) findViewById(R.id.file_view);
    }

    /**
     * Create file process
     *
     * @param view
     */
    public void onCreateFileClick(View view) {
        FileOutputStream fos = null;
        try {
            // *** POINT 1 *** Files must be created in application directory.
            // *** POINT 2 *** The access privilege of file must be set to read
            // only to other applications.
            // (MODE_WORLD_READABLE is deprecated API Level 17,
            // don't use this mode as much as possible and exchange data by using
            // ContentProvider().)
```

(continues on next page)

(continued from previous page)

```
            fos = openFileOutput(FILE_NAME, MODE_WORLD_READABLE);

            // *** POINT 3 *** Sensitive information must not be stored.
            // *** POINT 4 *** Regarding the information to be stored in files,
            // handle file data carefully and securely.
            // Omitted, since this is a sample. Please refer to
            // "3.2 Handling Input Data Carefully and Securely."
            fos.write(new String("Not sensitive information (Public File Activity)\
↪n").getBytes());
        } catch (FileNotFoundException e) {
            mFileView.setText(R.string.file_view);
        } catch (IOException e) {
            android.util.Log.e("PublicFileActivity",
                               "failed to read file");
        } finally {
            if (fos != null) {
                try {
                    fos.close();
                } catch (IOException e) {
                    android.util.Log.e("PublicFileActivity",
                                       "failed to close file");
                }
            }
        }

        finish();
    }

    /**
     * Read file process
     *
     * @param view
     */
    public void onReadFileClick(View view) {
        FileInputStream fis = null;
        try {
            fis = openFileInput(FILE_NAME);

            byte[] data = new byte[(int) fis.getChannel().size()];

            fis.read(data);

            String str = new String(data);

            mFileView.setText(str);
        } catch (FileNotFoundException e) {
            mFileView.setText(R.string.file_view);
        } catch (IOException e) {
            android.util.Log.e("PublicFileActivity",
                               "failed to read file");
        } finally {
            if (fis != null) {
                try {
                    fis.close();
                } catch (IOException e) {
                    android.util.Log.e("PublicFileActivity",
```

(continues on next page)

(continued from previous page)

```
                                        "failed to close file");
                }
            }
        }
    }

    /**
     * Delete file process
     *
     * @param view
     */
    public void onDeleteFileClick(View view) {

        File file = new File(this.getFilesDir() + "/" + FILE_NAME);
        file.delete();

        mFileView.setText(R.string.file_view);
    }
}
```

PublicUserActivity.java

```
/*
 * Copyright (C) 2012-2026 Japan Smartphone Security Association
 *
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 *
 *      http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */

package org.jssec.android.file.publicuser.readonly;

import java.io.File;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.IOException;

import android.app.Activity;
import android.content.ActivityNotFoundException;
import android.content.Context;
import android.content.Intent;
import android.content.pm.PackageManager.NameNotFoundException;
import android.os.Bundle;
import android.view.View;
import android.widget.TextView;

public class PublicUserActivity extends Activity {
```

(continues on next page)

(continued from previous page)

```
    private TextView mFileView;

    private static final String TARGET_PACKAGE =
            "org.jssec.android.file.publicfile.readonly";
    private static final String TARGET_CLASS =
            "org.jssec.android.file.publicfile.readonly.PublicFileActivity";

    private static final String FILE_NAME = "public_file.dat";

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.user);
        mFileView = (TextView) findViewById(R.id.file_view);
    }

    private void callFileActivity() {
        Intent intent = new Intent();
        intent.setClassName(TARGET_PACKAGE, TARGET_CLASS);

        try {
            startActivity(intent);
        } catch (ActivityNotFoundException e) {
            mFileView.setText("(File Activity does not exist)");
        }
    }

    /**
     * Call file Activity process
     *
     * @param view
     */
    public void onCallFileActivityClick(View view) {
        callFileActivity();
    }

    /**
     * Read file process
     *
     * @param view
     */
    public void onReadFileClick(View view) {
        FileInputStream fis = null;
        try {
            File file = new File(getFilesPath(FILE_NAME));
            fis = new FileInputStream(file);

            byte[] data = new byte[(int) fis.getChannel().size()];

            fis.read(data);

            // *** POINT 4 *** Regarding the information to be stored in files,
            // handle file data carefully and securely.
            // Omitted, since this is a sample. Please refer to
            // "3.2 Handling Input Data Carefully and Securely."
            String str = new String(data);
```

(continues on next page)

(continued from previous page)

```
            mFileView.setText(str);
        } catch (FileNotFoundException e) {
            android.util.Log.e("PublicUserActivity", "no file");
        } catch (IOException e) {
            android.util.Log.e("PublicUserActivity", "failed to read file");
        } finally {
            if (fis != null) {
                try {
                    fis.close();
                } catch (IOException e) {
                    android.util.Log.e("PublicUserActivity",
                                       "failed to close file");
                }
            }
        }
    }

    /**
     * Rewrite file process
     *
     * @param view
     */
    public void onWriteFileClick(View view) {
        FileOutputStream fos = null;
        boolean exception = false;
        try {
            File file = new File(getFilesPath(FILE_NAME));
            // Fail to write in. FileNotFoundException occurs.
            fos = new FileOutputStream(file, true);

            fos.write(new String("Not sensitive information (Public User Activity)\
↪n").getBytes());
        } catch (IOException e) {
            mFileView.setText(e.getMessage());
            exception = true;
        } finally {
            if (fos != null) {
                try {
                    fos.close();
                } catch (IOException e) {
                    exception = true;
                }
            }
        }

        if (!exception)
            callFileActivity();
    }

    private String getFilesPath(String filename) {
        String path = "";

        try {
            Context ctx = createPackageContext(TARGET_PACKAGE,
                                               Context.CONTEXT_RESTRICTED);
```

(continues on next page)

(continued from previous page)

```
            File file = new File(ctx.getFilesDir(), filename);
            path = file.getPath();
        } catch (NameNotFoundException e) {
            android.util.Log.e("PublicUserActivity", "no file");
        }
        return path;
    }
}
```

4.6.1.3 Using Public Read/Write Files

This is the usage of the file which permits read-write access to unspecified large number of application.

Unspecified large number of application can read and write, means that needless to say. Malware can also read and write, so the credibility and safety of data will be never guaranteed. In addition, even in case of not malicious intention, data format in file or timing to write in cannot be controlled. So this type of file is almost not practical in terms of functionality.

As above, it's impossible to use read-write files safely from both security and application designing points of view, so using read-write files should be avoided.

Point:

1. Must not create files that be allowed to read/write access from other applications.

4.6.1.4 Using Eternal Memory (Read Write Public) Files

This is the case when storing files in an external memory like SD card. It's supposed to be used when storing comparatively huge information (placing file which was downloaded from Web), or when bring out the information to outside (backup etc.).

"External memory file (Read Write public)" has the equal characteristics with "Read Write public file" to unspecified large number of applications. In addition, it has the equal characteristics with "Read Write public file" to applications which declares to use android.permission.WRITE_EXTERNAL_STORAGE Permission. So, the usage of "External memory file (Read Write public) file" should be minimized as less as possible.

A Backup file is most probably created in an external memory device as Android application's customary practice. However, as mentioned as above, files in an external memory have the risk that is tampered/deleted by other applications including malware. Hence, in applications which output backup, some contrivances to minimize risks in terms of application spec or designing like displaying a caution "Copy Backup files to the safety location like PC etc., a.s.a.p.", are necessary.

Because the filtered view for external storage (see "4.6.3.5. *External storage access for Android 10 and later*") is used as the default in Android 10 (API level 29), the following sample code (user side) does not run. However, the manifest attribute requestLegacyExternalStorage can be set to temporarily opt out of the scoped storage function. This is used only for temporary applications before the app is fully compatible or before app testing, and its use in the release version and other versions is not allowed. In the next major platform release, it is expected that scoped storage will be required in all apps regardless of the target SDK level.

Points:

1. Sensitive information must not be stored.
2. Files must be stored in the unique directory per application.
3. Regarding the information to be stored in files, handle file data carefully and securely.
4. Writing file by the requesting application should be prohibited as the specification.

Sample code for create

AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
          >

  <!-- declare android.permission.WRITE_EXTERNAL_STORAGE permission to write to␣
↪the external strage -->
  <!-- In Android 4.4 (API Level 19) and later, the application, which read/write␣
↪only files in its specific
       directories on external storage media, need not to require the permission␣
↪and it should declare
       the maxSdkVersion -->
  <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"
                   android:maxSdkVersion="18"/>

  <application
      android:icon="@drawable/ic_launcher"
      android:label="@string/app_name"
      android:allowBackup="false" >
    <activity
        android:name=".ExternalFileActivity"
        android:label="@string/app_name"
        android:exported="true" >
      <intent-filter>
        <action android:name="android.intent.action.MAIN" />

        <category android:name="android.intent.category.LAUNCHER" />
      </intent-filter>
    </activity>
  </application>
</manifest>
```

ExternalFileActivity.java

```
/*
 * Copyright (C) 2012-2026 Japan Smartphone Security Association
 *
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 *
 *      http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */

package org.jssec.android.file.externalfile;

import java.io.File;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.IOException;
```

(continues on next page)

(continued from previous page)

```
import android.app.Activity;
import android.os.Bundle;
import android.view.View;
import android.widget.TextView;

public class ExternalFileActivity extends Activity {

    private TextView mFileView;

    private static final String TARGET_TYPE = "external";

    private static final String FILE_NAME = "external_file.dat";

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.file);

        mFileView = (TextView) findViewById(R.id.file_view);
    }

    /**
     * Create file process
     *
     * @param view
     */
    public void onCreateFileClick(View view) {
        FileOutputStream fos = null;
        try {
            // *** POINT 1 *** Sensitive information must not be stored.
            // *** POINT 2 *** Files must be stored in the unique directory per
            // application.
            File file = new File(getExternalFilesDir(TARGET_TYPE), FILE_NAME);
            fos = new FileOutputStream(file, false);

            // *** POINT 3 *** Regarding the information to be stored in files,
            // handle file data carefully and securely.
            // Omitted, since this is a sample. Please refer to
            // "3.2 Handling Input Data Carefully and Securely."
            fos.write(new String("Non-Sensitive Information(ExternalFileActivity)\n
↪").getBytes());
        } catch (FileNotFoundException e) {
            mFileView.setText(R.string.file_view);
        } catch (IOException e) {
            android.util.Log.e("ExternalFileActivity",
                               "failed to read file");
        } finally {
            if (fos != null) {
                try {
                    fos.close();
                } catch (IOException e) {
                    android.util.Log.e("ExternalFileActivity",
                                       "failed to close file");
                }
            }
```

(continues on next page)

(continued from previous page)

```
        }

        finish();
    }

    /**
     * Read file process
     *
     * @param view
     */
    public void onReadFileClick(View view) {
        FileInputStream fis = null;
        try {
            File file = new File(getExternalFilesDir(TARGET_TYPE), FILE_NAME);
            fis = new FileInputStream(file);

            byte[] data = new byte[(int) fis.getChannel().size()];

            fis.read(data);

            // *** POINT 3 *** Regarding the information to be stored in files,
            // handle file data carefully and securely.
            // Omitted, since this is a sample. Please refer to
            // "3.2 Handling Input Data Carefully and Securely."
            String str = new String(data);

            mFileView.setText(str);
        } catch (FileNotFoundException e) {
            mFileView.setText(R.string.file_view);
        } catch (IOException e) {
            android.util.Log.e("ExternalFileActivity",
                               "failed to read file");
        } finally {
            if (fis != null) {
                try {
                    fis.close();
                } catch (IOException e) {
                    android.util.Log.e("ExternalFileActivity",
                                       "failed to close file");
                }
            }
        }
    }

    /**
     * Delete file process
     *
     * @param view
     */
    public void onDeleteFileClick(View view) {

        File file = new File(getExternalFilesDir(TARGET_TYPE), FILE_NAME);
        file.delete();

        mFileView.setText(R.string.file_view);
    }
```

(continues on next page)

(continued from previous page)

```
}
```

Sample code for use

AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
          >

  <queries>
    <package android:name="org.jssec.android.file.externalfile" />
  </queries>

  <!-- In Android 4.0.3 (API Level 14) and later, the permission for reading␣
↪external storages
       has been defined and the application should decalre that it requires the␣
↪permission.
       In fact in Android 4.4 (API Level 19) and later, that must be declared to␣
↪read other directories
       than the package specific directories. -->
  <uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE" />

  <application
      android:icon="@drawable/ic_launcher"
      android:label="@string/app_name"
      android:allowBackup="false" >

    <activity
        android:name=".ExternalUserActivity"
        android:label="@string/app_name"
        android:exported="true">
      <intent-filter>
        <action android:name="android.intent.action.MAIN" />

        <category android:name="android.intent.category.LAUNCHER" />
      </intent-filter>
    </activity>
  </application>
</manifest>
```

ExternalUserActivity.java

```
/*
 * Copyright (C) 2012-2026 Japan Smartphone Security Association
 *
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 *
 *      http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */
```

(continues on next page)

(continued from previous page)

```
package org.jssec.android.file.externaluser;

import java.io.File;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.IOException;

import android.Manifest;
import android.app.Activity;
import android.app.AlertDialog;
import android.content.ActivityNotFoundException;
import android.content.Context;
import android.content.DialogInterface;
import android.content.Intent;
import android.content.pm.PackageManager;
import android.content.pm.PackageManager.NameNotFoundException;
import android.os.Build;
import android.os.Bundle;
import android.view.View;
import android.widget.TextView;

public class ExternalUserActivity extends Activity {

    private TextView mFileView;

    private static final String TARGET_PACKAGE =
            "org.jssec.android.file.externalfile";
    private static final String TARGET_CLASS =
            "org.jssec.android.file.externalfile.ExternalFileActivity";
    private static final String TARGET_TYPE = "external";

    private static final String FILE_NAME = "external_file.dat";
    private final int MY_PERMISSIONS_REQUEST_READ_EXTERNAL_STORAGE = 1000;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.user);
        mFileView = (TextView) findViewById(R.id.file_view);
        // Android 6.0 (API level 23) or later requires dangerous permission
        // (in this case READ_EXTERNAL_STORAGE permission)
        // must be granted at runtime by user.
        // (Refer to “5.2.3.6. Modification to the Permission model Specifications
        // in Android versions 6.0 and later”)
        if (Build.VERSION.SDK_INT >= 23) {
            if (checkSelfPermission(Manifest.permission.READ_EXTERNAL_STORAGE)
                != PackageManager.PERMISSION_GRANTED) {
                requestPermissions(
                    new String[]{Manifest.permission.READ_EXTERNAL_STORAGE},
                    MY_PERMISSIONS_REQUEST_READ_EXTERNAL_STORAGE);
            }
        }
    }

    @Override
```

(continues on next page)

(continued from previous page)

```
    public void onRequestPermissionsResult(int requestCode,
                                           String permissions[],
                                           int[] grantResults) {
        if (requestCode == MY_PERMISSIONS_REQUEST_READ_EXTERNAL_STORAGE) {
            if (grantResults[0] != PackageManager.PERMISSION_GRANTED) {
                finish();
            }
        }
    }

    private void callFileActivity() {
        Intent intent = new Intent();
        intent.setClassName(TARGET_PACKAGE, TARGET_CLASS);

        try {
            startActivity(intent);
        } catch (ActivityNotFoundException e) {
            mFileView.setText("(File Activity does not exist)");
        }
    }

    /**
     * Call file Activity process
     *
     * @param view
     */
    public void onCallFileActivityClick(View view) {
        callFileActivity();
    }

    /**
     * Read file process
     *
     * @param view
     */
    public void onReadFileClick(View view) {
        FileInputStream fis = null;
        try {
            File file = new File(getFilesPath(FILE_NAME));
            fis = new FileInputStream(file);

            byte[] data = new byte[(int) fis.getChannel().size()];

            fis.read(data);

            // *** POINT 3 *** Regarding the information to be stored in files,
            // handle file data carefully and securely.
            // Omitted, since this is a sample. Please refer to
            // "3.2 Handling Input Data Carefully and Securely."
            String str = new String(data);

            mFileView.setText(str);
        } catch (FileNotFoundException e) {
            mFileView.setText(R.string.file_view);
        } catch (IOException e) {
            android.util.Log.e("ExternalUserActivity",
```

(continues on next page)

(continued from previous page)

```
                                "failed to read file");
        } finally {
            if (fis != null) {
                try {
                    fis.close();
                } catch (IOException e) {
                    android.util.Log.e("ExternalUserActivity",
                                       "failed to close file");
                }
            }
        }
    }

    /**
     * Rewrite file process
     *
     * @param view
     */
    public void onWriteFileClick(View view) {

        // *** POINT 4 *** Writing file by the requesting application should be
        // prohibited as the specification.
        // Application should be designed supposing malicious application may
        // overwrite or delete file.

        final AlertDialog.Builder alertDialogBuilder =
                new AlertDialog.Builder(this);
        alertDialogBuilder.setTitle("POINT 4");
        alertDialogBuilder.setMessage("Do not write in calling appllication.");
        alertDialogBuilder.setPositiveButton("OK",
                new DialogInterface.OnClickListener() {

                @Override
                public void onClick(DialogInterface dialog, int which) {
                    callFileActivity();
                }
        });

        alertDialogBuilder.create().show();

    }

    private String getFilesPath(String filename) {
        String path = "";

        try {
            Context ctx = createPackageContext(TARGET_PACKAGE,
                                               Context.CONTEXT_IGNORE_SECURITY);
            File file = new File(ctx.getExternalFilesDir(TARGET_TYPE), filename);
            path = file.getPath();
        } catch (NameNotFoundException e) {
            android.util.Log.e("ExternalUserActivity", "no file");
        }
        return path;
    }
}
```

4.6.2 Rule Book

Handling files follow the rules below.

1. *File Must Be Created as a Private File in Principle (Required)*
2. *Must Not Create Files that Be Allowed to Read/Write Access from Other Applications (Required)*
3. *Using Files Stored in External Device (e.g. SD Card) Should Be Requisite Minimum (Required)*
4. *Application Should Be Designed Considering the Scope of File (Required)*

4.6.2.1 File Must Be Created as a Private File in Principle (Required)

As mentioned in "4.6. *Handling Files*" and "4.6.1.1. *Using Private Files*," regardless of the contents of the information to be stored, files should be set private, in principle. From Android security designing point of view, exchanging information and its access control should be done in Android system like Content Provider and Service, etc., and in case there's a reason that is impossible, it should be considered to be substituted by file access permission as alternative method.

Please refer to sample code of each file type and following rule items.

4.6.2.2 Must Not Create Files that Be Allowed to Read/Write Access from Other Applications (Required)

As mentioned in "4.6.1.3. *Using Public Read/Write Files*," when permitting other applications to read/write files, information stored in files cannot be controlled. So, sharing information by using read/write public files should not be considered from both security and function/designing points of view.

4.6.2.3 Using Files Stored in External Device (e.g. SD Card) Should Be Requisite Minimum (Required)

As mentioned in "4.6.1.4. *Using Eternal Memory (Read Write Public) Files*," storing files in external memory device like SD card, leads to holding the potential problems from security and functional points of view. On the other hand, SD card can handle files which have longer scope, compared with application directory, and this is the only one storage that can be always used to bring out the data to outside of application. So, there may be many cases that cannot help using it, depends on application's spec.

When storing files in external memory device, considering unspecified large number of applications and users can read/write/delete files, so it's necessary that application is designed considering the points as per below as well as the points mentioned in sample code.

- Sensitive information should not be saved in a file of external memory device, in principle.
- In case sensitive information is saved in a file of external memory device, it should be encrypted.
- In case saving in a file of external memory device information that will be trouble if it's tampered by other application or users, it should be saved with electrical signature.
- When reading in files in external memory device, use data after verifying the safety of data to read in.
- Application should be designed supposing that files in external memory device can be always deleted.

Please refer to "4.6.2.4. *Application Should Be Designed Considering the Scope of File (Required)*."

4.6.2.4 Application Should Be Designed Considering the Scope of File (Required)

Data saved in application directory is deleted by the following user operations. It's consistent with the application's scope, and it's distinctive that it's shorter than the scope of application.

- Uninstalling application.

- Delete data and cache of each application. ("Setting" > "Apps" > "select target application")

Files that were saved in external memory device like SD card, it's distinctive that the scope of the file is longer than the scope of the application. In addition, the following situations are also necessary to be considered.

- File deletion by user
- Pick off/replace/unmount SD card
- File deletion by Malware

As mentioned above, since scope of files are different depends on the file saving location, not only from the viewpoint to protect sensitive information, but also form view point to achieve the right behavior as application, it's necessary to select the file save location.

4.6.3 Advanced Topics

4.6.3.1 File Sharing Through File Descriptor

There is a method to share files through file descriptor, not letting other applications access to public files. This method can be used in Content Provider and in Service. Opponent application can read/write files through file descriptors which are got by opening private files in Content Provider or in Service.

Comparison between the file sharing method of direct access by other applications and the file sharing method via file descriptor, is as per below Table 4.6.2. Variation of access permission and range of applications that are permitted to access, can be considered as merits. Especially, from security point of view, this is a great merit that, applicaions that are permitted to accesss can be controlled in detail.

Table 4.6.2: Comparison of inter-application file sharing method

File sharing method	Variation or access permission setting	Range of applications that are permitted to access
File sharing that permits other applications to access files directry	• Read in • Write in • Read in + Write in	Give all application access permissions equally
File sharing through file descriptor	• Read in • Write in • Only add • Read in + Write in • Read in + Only add	Can control whether to give access permission or not, to application which try to access indivisually and temporarily, to Content provider or Service

This is common in both of above file sharing methods, when giving write permission for files to other applications, integrity of file contents are difficult to be guaranteed. When several applications write in in parallel, there's a risk that data structure of file contents are destroyed, and application doesn't work normally. So, in sharing files with other applications, giving only read only permission is preferable.

Herein below an implementation example of file sharing by Content Provider and its sample code, are published.

Point

1. The source application is In house application, so sensitive information can be saved.
2. Even if it's a result from In house only Content Provider application, verify the safety of the result data.

InhouseProvider.java

```
/*
 * Copyright (C) 2012-2026 Japan Smartphone Security Association
 *
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
```

(continues on next page)

(continued from previous page)

```
 * You may obtain a copy of the License at
 *
 *      http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */

package org.jssec.android.file.inhouseprovider;

import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.IOException;

import org.jssec.android.shared.SigPerm;
import org.jssec.android.shared.Utils;

import android.content.ContentProvider;
import android.content.ContentValues;
import android.content.Context;
import android.database.Cursor;
import android.net.Uri;
import android.os.ParcelFileDescriptor;

public class InhouseProvider extends ContentProvider {

    private static final String FILENAME = "sensitive.txt";

    // In-house signature permission
    private static final String MY_PERMISSION =
            "org.jssec.android.file.inhouseprovider.MY_PERMISSION";

    // In-house certificate hash value
    private static String sMyCertHash = null;

    private static String myCertHash(Context context) {
        if (sMyCertHash == null) {
            if (Utils.isDebuggable(context)) {
                // Certificate hash value of  debug.keystore "androiddebugkey"
                sMyCertHash = "0EFB7236 328348A9 89718BAD DF57F544 D5CCB4AE␣
↪B9DB34BC 1E29DD26 F77C8255";
            } else {
                // Certificate hash value of  keystore "my company key"
                sMyCertHash = "D397D343 A5CBC10F 4EDDEB7C A10062DE 5690984F␣
↪1FB9E88B D7B3A7C2 42E142CA";
            }
        }
        return sMyCertHash;
    }

    @Override
    public boolean onCreate() {
```

(continues on next page)

(continued from previous page)

```
        File dir = getContext().getFilesDir();
        FileOutputStream fos = null;
        try {
            fos = new FileOutputStream(new File(dir, FILENAME));
            // *** POINT 1 *** The source application is In house application,
            // so sensitive information can be saved.
            fos.write(new String("Sensitive information").getBytes());

        } catch (IOException e) {
            android.util.Log.e("InhouseProvider", "failed to read file");
        } finally {
            try {
                fos.close();
            } catch (IOException e) {
                android.util.Log.e("InhouseProvider", "failed to close file");
            }
        }

        return true;
    }

    @Override
    public ParcelFileDescriptor openFile(Uri uri, String mode)
        throws FileNotFoundException {

        // Verify that in-house-defined signature permission is defined by
        // in-house application.
        if (!SigPerm
            .test(getContext(), MY_PERMISSION, myCertHash(getContext()))) {
            throw new SecurityException("In-house-defined signature permission is
↪not defined by in-house application.");
        }

        File dir = getContext().getFilesDir();
        File file = new File(dir, FILENAME);

        // Always return read-only, since this is sample
        int modeBits = ParcelFileDescriptor.MODE_READ_ONLY;
        return ParcelFileDescriptor.open(file, modeBits);
    }

    @Override
    public String getType(Uri uri) {
        return "";
    }

    @Override
    public Cursor query(Uri uri, String[] projection, String selection,
                        String[] selectionArgs, String sortOrder) {
        return null;
    }

    @Override
    public Uri insert(Uri uri, ContentValues values) {
        return null;
    }
```

(continues on next page)

(continued from previous page)

```
    @Override
    public int update(Uri uri, ContentValues values, String selection,
                      String[] selectionArgs) {
        return 0;
    }

    @Override
    public int delete(Uri uri, String selection, String[] selectionArgs) {
        return 0;
    }
}
```

InhouseUserActivity.java

```
/*
 * Copyright (C) 2012-2026 Japan Smartphone Security Association
 *
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 *
 *      http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */

package org.jssec.android.file.inhouseprovideruser;

import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.IOException;

import org.jssec.android.shared.PkgCert;
import org.jssec.android.shared.SigPerm;
import org.jssec.android.shared.Utils;

import android.app.Activity;
import android.content.Context;
import android.content.pm.PackageManager;
import android.content.pm.ProviderInfo;
import android.net.Uri;
import android.os.Bundle;
import android.os.ParcelFileDescriptor;
import android.view.View;
import android.widget.TextView;

public class InhouseUserActivity extends Activity {

    // Content Provider information of destination (requested provider)
    private static final String AUTHORITY =
            "org.jssec.android.file.inhouseprovider";
```

(continues on next page)

(continued from previous page)

```
    // In-house signature permission
    private static final String MY_PERMISSION =
            "org.jssec.android.file.inhouseprovider.MY_PERMISSION";

    // In-house certificate hash value
    private static String sMyCertHash = null;

    private static String myCertHash(Context context) {

        if (sMyCertHash == null) {
            if (Utils.isDebuggable(context)) {
                // Certificate hash value of debug.keystore "androiddebugkey"
                sMyCertHash = "0EFB7236 328348A9 89718BAD DF57F544 D5CCB4AE␣
↪B9DB34BC 1E29DD26 F77C8255";
            } else {
                // Certificate hash value of keystore "my company key"
                sMyCertHash = "D397D343 A5CBC10F 4EDDEB7C A10062DE 5690984F␣
↪1FB9E88B D7B3A7C2 42E142CA";
            }
        }
        return sMyCertHash;
    }

    // Get package name of destination (requested) content provider.
    private static String providerPkgname(Context context, String authority) {
        String pkgname = null;
        PackageManager pm = context.getPackageManager();
        ProviderInfo pi = pm.resolveContentProvider(authority, 0);
        if (pi != null)
            pkgname = pi.packageName;
        return pkgname;
    }

    public void onReadFileClick(View view) {

        logLine("[ReadFile]");

        // Verify that in-house-defined signature permission is defined by
        // in-house application.
        if (!SigPerm.test(this, MY_PERMISSION, myCertHash(this))) {
            logLine("  In-house-defined signature permission is not defined by in-
↪house application.");
            return;
        }

        // Verify that the certificate of destination (requested) content provider
        // application is in-house certificate.
        String pkgname = providerPkgname(this, AUTHORITY);
        if (!PkgCert.test(this, pkgname, myCertHash(this))) {
            logLine("  Destination (Requested) Content Provider is not in-house␣
↪application.");
            return;
        }

        // Only the information which can be disclosed to in-house only content
        // provider application, can be included in a request.
```

(continues on next page)

(continued from previous page)

```
        ParcelFileDescriptor pfd = null;
        try {
            pfd = getContentResolver()
                    .openFileDescriptor(Uri.parse("content://" + AUTHORITY), "r");
        } catch (FileNotFoundException e) {
            android.util.Log.e("InhouseUserActivity", "no file");
        }

        if (pfd != null) {
            FileInputStream fis = new FileInputStream(pfd.getFileDescriptor());

            if (fis != null) {
                try {
                    byte[] buf = new byte[(int) fis.getChannel().size()];
                    fis.read(buf);
                    // *** POINT 2 *** Handle received result data carefully and
                    // securely, even though the data came from in-house
                    // applications.
                    // Omitted, since this is a sample. Please refer to
                    // "3.2 Handling Input Data Carefully and Securely."
                    logLine(new String(buf));
                } catch (IOException e) {
                    android.util.Log.e("InhouseUserActivity",
                                       "failed to read file");
                } finally {
                    try {
                        fis.close();
                    } catch (IOException e) {
                        android.util.Log.e("ExternalFileActivity",
                                           "failed to close file");
                    }
                }
            }
            try {
                pfd.close();
            } catch (IOException e) {
                android.util.Log.e("ExternalFileActivity",
                                   "failed to close file descriptor");
            }

        } else {
            logLine("  null file descriptor");
        }
    }

    private TextView mLogView;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        mLogView = (TextView) findViewById(R.id.logview);
    }

    private void logLine(String line) {
        mLogView.append(line);
```

(continues on next page)

(continued from previous page)

```
        mLogView.append("\n");
    }
}
```

4.6.3.2 Access Permission Setting for the Directory

Herein above, security considerations are explained, focusing on files. It's also necessary to consider the security for directory which is a file container. Herein below, security considerations of access permission setting for directory are explained.

In Android, there are some methods to get/create subdirectory in application directory. The major ones are as per below Table 4.6.3.

Table 4.6.3: Methods to get/create subdirectory in application directory

	Specify access permission to other applications	Deletion by User
Context#getFilesDir()	Impossible (Only execution permission)	"Setting" > "Apps" > select target application > "Clear data"
Context#"getCacheDir()	Impossible (Only execution permission)	"Setting" > "Apps" > select target application > "Clear cache" (It can be deleted by "Clear data," too.)
Context#getDir(String name, int mode)	modes MODE_PRIVATE, MODE_WORLD_READABLE or MODE_WORLD_WRITEABLE can be specified as a MODE	"Setting" > "Apps" > select target application > "Clear data"

Here especially what needs to pay attention is access permission setting by Context#getDir(). As explained in file creation, basically directory also should be set private from the security designing point of view. When sharing information depends on access permission setting, there may be an unexpected side effect, so other methods should be taken as information sharing.

MODE_WORLD_READABLE

This is a flag to give all applications read-only permission to directory. So all applications can get file list and individual file attribute information in the directory. Because secret files may not be placed in these directories, in general this flag must not be used.[23]

MODE_WORLD_WRITEABLE

This flag gives other applications write permission to directory. All applications can create/move[24]/rename/delete files in the directory. These operations has no relation with access permission setting (Read/Write/Execute) of file itself, so it's necessary to pay attention that operations can be done only with write permission to directory. This flag allows other apps to delete or replace files arbitrarily, so in general it must not be used.[23]

Regarding Table 4.6.3 "Deletion by User", refer to "4.6.2.4. *Application Should Be Designed Considering the Scope of File (Required)*."

4.6.3.3 Access Permission Setting for Shared Preference and Database File

Shared Preference and database also consist of files. Regarding access permission setting what are explained for files are applied here. Therefore, both Shared Preference and database, should be created as private files same like files, and sharing contents should be achieved by the Android's inter-application linkage system.

[23] MODE_WORLD_READABLE and MODE_WORLD_WRITEABLE are deprecated in API Level17 and later versions, and in API Level 24 and later versions their use will trigger a security exception.

[24] Files cannot be moved over mount point (e.g. from internal storage to external storage). Therefore, moving the protected files from internal storage to external storage cannot be happened.

Herein below, the usage example of Shared Preference is shown. Shared Preference is crated as private file by MODE_PRIVATE.

Example of setting access restriction to Shared Preference file.

import android.content.SharedPreferences; import android.content.SharedPreferences.Editor;

```
import android.content.SharedPreferences;
import android.content.SharedPreferences.Editor;
... Abbreviation ...
// Get Shared Preference.
// (If there's no Shared Preference, it's to be created.)
// Point: Basically, specify MODE_PRIVATE mode.
SharedPreferences preference = getSharedPreferences(
    PREFERENCE_FILE_NAME, MODE_PRIVATE);
// Example of writing preference which value is charcter string.
Editor editor = preference.edit();
// key:"prep_key", value:"prep_value"
editor.putString("prep_key", "prep_value");
editor.commit();
```

Please refer to "4.5. *Using SQLite*" for database.

4.6.3.4 Internal storage access for Android 10 and later

There are no changes to the internal storage specifications from Android 9 and earlier. The internal storage is originally separated for each app, and it is designed so that apps cannot access data from other apps.

Your app's internal storage area (such as /data/data/<package_name>/files/) can be freely read and written without permission using Context file I/O methods such as openFileOutput() and FileOutputStream. On the other hand, access to the internal storage area of other apps is completely prohibited on Android 9 and earlier, as well as Android 10 and later. Internal storage data is deleted when the app is uninstalled.

As such, the internal storage access specifications have not changed between Android 9 and 10, and the principles of "only your own app can access it" and "other apps' data cannot be accessed" have always been maintained.

Table 4.6.4: Internal storage access for Android 10 and later

Use	Access methods	Method to get a path	Actual path	Required permissions
Own app's private data	Context File I/O Methods File API	context.getFilesDir()	/data/data/<package_name>/files/	Not required
Other app's private data	Normally not accessible	N/A	N/A	N/A

4.6.3.5 External storage access for Android 10 and later

Android 10 and later significantly strengthened the security and privacy of accessing external storage (including SD cards). To better protect user privacy, access to data from other apps has been strictly restricted.

Up until Android 9, an app could access its own data in external storage without permission, but accessing other apps' data or their secondary storage required the READ_EXTERNAL_STORAGE or WRITE_EXTERNAL_STORAGE permissions. The File API also allowed access to other apps' data by directly specifying the path.

With the introduction of Scoped Storage in Android 10 and later, access to other apps' specific data and external storage areas is essentially prohibited. This significantly limits access to data stored by other apps, and it is no longer possible to access data even by directly specifying a path using the File API, etc. There are no changes to the

method of accessing or obtaining the path to your own app's external storage area (such as /storage/emulated/0/Android/data/<package_name>/files/), and you can still read and write freely without requiring permission.

Below is a summary of the differences in external storage access between Android 9 and 10 and later.

Table 4.6.5: External storage access in Android 9

Use	Access methods	Method to get a path	Actual path	Required permissions
Own app's specific data	File API	context.getExternalFilesDir(null) context.getExternalFilesDirs(null)[1]	/storage/emulated/0/Android/data/<package_name>/files/ /storage/extSdCard/ /storage/sdcard1/	Not required
Other app's specific data	File API	Specify direct path	/storage/emulated/0/Android/data/<other_package_name>/files/ /storage/extSdCard/ /storage/sdcard1/	READ_EXTERNAL_STORAGE WRITE_EXTERNAL_STORAGE

Table 4.6.6: External storage access for Android 10 and later

Use	Access methods	Method to get a path	Actual path	Required permissions
Own app's specific data	File API	context.getExternalFilesDir(null) context.getExternalFilesDirs(null)[1]	/storage/emulated/0/Android/data/<package_name>/files/ /storage/<external_sd>/Android/data/<package_name>/files/	Not Required
Other app's specific data	Normally not accessible	N/A	N/A	N/A

4.6.3.6 Shared storage access for Android 10 and later

In Android 10 and later, security and privacy regarding access to shared storage have been strengthened. As a result, direct access using the conventional path-based File API is not recommended, and it is now standard to access files indirectly using the MediaStore API or Storage Access Framework (SAF).

By using the MediaStore API or SAF, your app can read and write files that it stores and manages itself without requiring special permissions. This allows you to freely manipulate app-specific data. On the other hand, to access files stored by other apps (such as user photos or files in the Downloads folder), you need read permissions (e.g., READ_MEDIA_IMAGES or READ_EXTERNAL_STORAGE). Note that you are only allowed to read this data from other apps; you cannot directly write to or delete it.

Starting with Android 13, the OS's standard "Photo Picker" feature has been added to allow apps to access existing media files, such as images and videos, on their devices. This feature is limited to reading existing files selected by the user. Photo Picker does not require storage permissions and can only temporarily access image and video files selected by the user, so it is officially recommended by Google from a privacy perspective. Files obtained via Photo Picker are also read-only and cannot be written to.

In summary, in Android 9 and earlier, files could be accessed directly using the File API or directory paths, but in Android 10 and later, to enhance security, files must be accessed indirectly using the MediaStore API or SAF. While your own app's files can be read and written without permission, other apps' files require permission and can only be read; they cannot be written. In Android 13 and later, it is recommended that you use the Photo Picker if you want the user to select existing media files such as images and videos and pass them to your app. You should continue to use the MediaStore API and SAF for your app's file management and new file saving.

Table 4.6.7: Shared Storage Access in Android 9

Use	Access methods	Method to get a path	Actual path	Required permissions
Own app's shared data	File API	Environment.getExternalStoragePublicDirectory() Environment.DIRECTORY_MUSIC Environment.DIRECTORY_PICTURES Environment.DIRECTORY_MOVIES etc.	/storage/emulated/0/Music /storage/emulated/0/Pictures /storage/emulated/0/Movies etc.	READ_EXTERNAL_STORAGE WRITE_EXTERNAL_STORAGE
Other app's shared data	File API	Environment.getExternalStoragePublicDirectory() Environment.DIRECTORY_MUSIC Environment.DIRECTORY_PICTURES Environment.DIRECTORY_MOVIES etc.	/storage/emulated/0/Music /storage/emulated/0/Pictures /storage/emulated/0/Movies etc.	READ_EXTERNAL_STORAGE WRITE_EXTERNAL_STORAGE

Table 4.6.8: Shared storage access for Android 10 and later

Use	Access methods	Method to get a path	Actual path	Required permissions
Own app's shared data	MediaStore API Storage Access Framework	ContentResolver SAF	URI obtained by ContentResolver URI obtained by SAF	Not Required
Other app's shared data	MediaStore API Storage Access Framework	ContentResolver SAF	URI obtained by ContentResolver URI obtained by SAF	Storage read permission Requires explicit user consent

Table 4.6.9: Shared storage access for Android 13 and later

Use	Access methods	Method to get a path	Actual path	Required permissions
Own app's shared data	MediaStore API Storage Access Framework	ContentResolver SAF	URI obtained by ContentResolver URI obtained by SAF	Not Required
Other app's shared data	MediaStore API Storage Access Framework	ContentResolver SAF	URI obtained by ContentResolver URI obtained by SAF	Storage read permission Requires explicit user consent
Existing image and video files	Photo Picker API	Photo Picker API	URI obtained from Photo Picker API	Not Required

Own app's shared data

The following sample code uses the MediaStore API to show how to access your app's shared data. Specifically, create a text file named "example.txt" in the Documents folder and write the text "Hello, World!" to it.

```
import android.content.ContentValues
import android.os.Build
import android.os.Bundle
import android.provider.MediaStore
import androidx.annotation.RequiresApi
import androidx.appcompat.app.AppCompatActivity
import java.io.OutputStream

@RequiresApi(Build.VERSION_CODES.Q)
class MainActivity : AppCompatActivity() {

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        writeFileToMediaStore()
    }

    private fun writeFileToMediaStore() {
        // Set ContentValues for saving to MediaStore
        val contentValues = ContentValues().apply {
            put(MediaStore.MediaColumns.DISPLAY_NAME, "example.txt")
            put(MediaStore.MediaColumns.MIME_TYPE, "text/plain")
            put(MediaStore.MediaColumns.RELATIVE_PATH, "Documents/")
        }

        // Insert a new item into MediaStore
        val uri = contentResolver.insert(MediaStore.Files.getContentUri("external
↪"), contentValues)

        // Use the URI to write data to the file
        uri?.let {
            val outputStream: OutputStream? = contentResolver.openOutputStream(it)
            outputStream?.use {
                it.write("Hello, World!".toByteArray())
            }
        }
    }
}
```

Other app's shared data

To access shared data belonging to other apps, you can use the MediaStore API to read media files such as images, videos, music, and documents. However, this requires storage-related permissions, and the types of permissions required and their behavior vary by Android version. It should also be noted that shared data belonging to other apps is read-only; direct writing or deletion is not possible.

Below is a summary of the permission requirements for accessing shared data belonging to other apps using the MediaStore API, along with differences between Android versions.

Table 4.6.10: Storage read permissions for Android 10 and later

Media Type	Use	Access methods	Required permissions
Image	Other app's image data	MediaStore.Images.Media.EXTERNAL_CONTENT_URI	READ_EXTERNAL_STORAGE
Music	Other app's music data	MediaStore.Audio.Media.EXTERNAL_CONTENT_URI	READ_EXTERNAL_STORAGE
Video	Other app's video data	MediaStore.Video.Media.EXTERNAL_CONTENT_URI	READ_EXTERNAL_STORAGE
Document	Other app's document data	MediaStore.Files.getContentUri("external")	READ_EXTERNAL_STORAGE

Table 4.6.11: Storage read permissions for Android 13 and later

Media Type	Use	Access methods	Required permissions
Image	Other app's image data	MediaStore.Images.Media.EXTERNAL_CONTENT_URI	READ_MEDIA_IMAGES
Music	Other app's music data	MediaStore.Audio.Media.EXTERNAL_CONTENT_URI	READ_MEDIA_AUDIO
Video	Other app's video data	MediaStore.Video.Media.EXTERNAL_CONTENT_URI	READ_MEDIA_VIDEO
Document	Other app's document data	MediaStore.Files.getContentUri("external")	READ_EXTERNAL_STORAGE

Table 4.6.12: Storage read permissions for Android 14 and later

Media Type	Use	Access methods	Required permissions
Image	Other app's image data	MediaStore.Images.Media.EXTERNAL_CONTENT_URI	READ_MEDIA_IMAGES
Music	Other app's music data	MediaStore.Audio.Media.EXTERNAL_CONTENT_URI	READ_MEDIA_AUDIO
Video	Other app's video data	MediaStore.Video.Media.EXTERNAL_CONTENT_URI	READ_MEDIA_VIDEO
Document	Other app's document data	MediaStore.Files.getContentUri("external")	READ_EXTERNAL_STORAGE
User-selected visual media	User-selected image and video data	MediaStore.Images.Media.EXTERNAL_CONTENT_URI / MediaStore.Video.Media.EXTERNAL_CONTENT_URI	READ_MEDIA_VISUAL_USER_SELECTED

Below is a sample code using READ_MEDIA_VIDEO, which displays the title of a video file stored on the device on the loading screen.

```
<uses-permission android:name="android.permission.READ_MEDIA_VIDEO"/>
```

```
package com.example.myapplication

import android.content.pm.PackageManager
import android.os.Build
import android.os.Bundle
import android.provider.MediaStore
import androidx.activity.ComponentActivity
import androidx.activity.compose.setContent
import androidx.activity.result.contract.ActivityResultContracts
import androidx.activity.enableEdgeToEdge
```

(continues on next page)

(continued from previous page)

```
import androidx.compose.foundation.layout.fillMaxSize
import androidx.compose.foundation.layout.padding
import androidx.compose.foundation.lazy.LazyColumn
import androidx.compose.foundation.lazy.items
import androidx.compose.material3.Scaffold
import androidx.compose.material3.Text
import androidx.compose.runtime.*
import androidx.compose.ui.Modifier
import androidx.compose.ui.tooling.preview.Preview
import androidx.core.content.ContextCompat
import com.example.myapplication.ui.theme.MyApplicationTheme

class MainActivity : ComponentActivity() {
    private val requestPermissionLauncher = registerForActivityResult(
        ActivityResultContracts.RequestPermission()
    ) { isGranted: Boolean ->
        if (isGranted) {
            loadVideoFiles()
        } else {
            // Process when permission is denied
        }
    }

    private var videoFiles by mutableStateOf(listOf<String>())

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        enableEdgeToEdge()
        setContent {
            MyApplicationTheme {
                Scaffold(modifier = Modifier.fillMaxSize()) { innerPadding ->
                    VideoList(
                        videoFiles = videoFiles,
                        modifier = Modifier.padding(innerPadding)
                    )
                }
            }
        }
        checkAndRequestPermission()
    }

    private fun checkAndRequestPermission() {
        if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.TIRAMISU) {
            when {
                ContextCompat.checkSelfPermission(
                    this,
                    android.Manifest.permission.READ_MEDIA_VIDEO
                ) == PackageManager.PERMISSION_GRANTED -> {
                    loadVideoFiles()
                }
                shouldShowRequestPermissionRationale(android.Manifest.permission.
↪READ_MEDIA_VIDEO) -> {
                    // Show explanation to the user
                }
                else -> {
                    requestPermissionLauncher.launch(android.Manifest.permission.
```

(continues on next page)

(continued from previous page)

```
↪READ_MEDIA_VIDEO)
                }
            }
        } else {
            // Fallback for older Android version devices
            loadVideoFiles()
        }
    }

    private fun loadVideoFiles() {
        val projection = arrayOf(MediaStore.Video.Media.TITLE)
        val cursor = contentResolver.query(
            MediaStore.Video.Media.EXTERNAL_CONTENT_URI,
            projection,
            null,
            null,
            null
        )

        cursor?.use {
            val titleIndex = it.getColumnIndexOrThrow(MediaStore.Video.Media.TITLE)
            val videoList = mutableListOf<String>()
            while (it.moveToNext()) {
                val title = it.getString(titleIndex)
                videoList.add(title)
            }
            videoFiles = videoList
        }
    }
}

@Composable
fun VideoList(videoFiles: List<String>, modifier: Modifier = Modifier) {
    LazyColumn(modifier = modifier) {
        items(videoFiles) { video ->
            Text(text = video)
        }
    }
}

@Preview(showBackground = true)
@Composable
fun VideoListPreview() {
    MyApplicationTheme {
        VideoList(videoFiles = listOf("Sample Video 1", "Sample Video 2"))
    }
}
```

Existing Image and Video Files

Starting with Android 13, apps are recommended to use the Photo Picker if they want to access existing image or video files. The Photo Picker allows users to explicitly select images or videos from the device or cloud, without requesting storage permissions. This is because users choose which files to pass to the app each time; apps do not automatically have access to all media files. Files obtained via the Photo Picker are only available temporarily; permanent access is not granted. The functionality provided by the Photo Picker is limited. If you have requirements that the Photo Picker cannot meet, such as a custom gallery UI or ongoing file management, you should consider using the MediaStore API or the Storage Access Framework (SAF).

Sample code using the Photo Picker is shown below. Pressing the "Select Image" button launches the Photo Picker, and the URI of the selected image file is displayed on the screen.

```
package com.example.myapplication

import android.net.Uri
import android.os.Bundle
import androidx.activity.ComponentActivity
import androidx.activity.compose.rememberLauncherForActivityResult
import androidx.activity.compose.setContent
import androidx.activity.result.contract.ActivityResultContracts
import androidx.activity.result.PickVisualMediaRequest
import androidx.activity.enableEdgeToEdge
import androidx.compose.foundation.layout.*
import androidx.compose.material3.*
import androidx.compose.runtime.*
import androidx.compose.ui.Modifier
import androidx.compose.ui.tooling.preview.Preview
import androidx.compose.ui.unit.dp
import com.example.myapplication.ui.theme.MyApplicationTheme

class MainActivity : ComponentActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        enableEdgeToEdge()
        setContent {
            MyApplicationTheme {
                Scaffold(modifier = Modifier.fillMaxSize()) { innerPadding ->
                    PhotoPickerDemo(modifier = Modifier.padding(innerPadding))
                }
            }
        }
    }
}

@Composable
fun PhotoPickerDemo(modifier: Modifier = Modifier) {
    var selectedUri by remember { mutableStateOf<Uri?>(null) }
    val launcher = rememberLauncherForActivityResult(
        ActivityResultContracts.PickVisualMedia()
    ) { uri: Uri? ->
        selectedUri = uri
    }

    Column(
        modifier = modifier
            .fillMaxSize()
            .padding(16.dp),
        verticalArrangement = Arrangement.spacedBy(16.dp)
    ) {
        Button(onClick = {
            launcher.launch(
                PickVisualMediaRequest(ActivityResultContracts.PickVisualMedia.
↪ImageOnly)
            )
        }) {
            Text("Select Image")
        }
```

(continues on next page)

(continued from previous page)

```
        if (selectedUri != null) {
            Text("Selected image URI: $selectedUri")
        }
    }
}

@Preview(showBackground = true)
@Composable
fun PhotoPickerDemoPreview() {
    MyApplicationTheme {
        PhotoPickerDemo()
    }
}
```

4.6.3.7 Enhanced Safety of DCL (Dynamic Code Loading)

Starting with Android 14, when dynamically loading code, it is mandatory to use the setReadOnly () method immediately after opening a file to be read, such as DEX, JAR, or APK, to make it read-only.

This is not only to prevent file conflicts, but also to reduce the risk that the read code itself will be tampered with and the application will be used improperly.

An implementation example using the setReadOnly() method is shown below.

```
File jar = new File(dexPath);
try (FileOutputStream os = new FileOutputStream(jar)) {
    // Set the file to read-only first to prevent race conditions
    jar.setReadOnly();
    // Then write the actual file content
} catch (IOException e) {
    Log.d("Log", e.toString());
}
PathClassLoader cl = new PathClassLoader(dexPath, getClass().getClassLoader());
```

If the setReadOnly() method is not used, the following error occurs when creating the PathClassLoader instance[25].

```
E  Attempt to load writable dex file: /data/user/0/com.example.test/cache/Test_dex.
↪jar
```

From the standpoint of security, the use of DCL itself is not recommended. As mentioned above, there is a risk of code injection and code tampering. The user is not aware of what code the application is loading internally. We believe that all responsibility for the behavior of the application belongs to the provider of the application. Therefore, if you are currently using DCL, you should consider alternative methods.

The official recommendation is to use the Android App Bundle as an alternative method, and from August 2021, the use of the Android App Bundle has become mandatory for all new applications in the Google Play Store[26].

Expansion of Protection Scope to Native Libraries in Android 17

In Android 17 and later, this DCL protection has been further strengthened, expanding the read-only requirement previously applied to DEX, JAR, etc., to native libraries (.so files). Specifically, native files loaded dynamically using System.load() or similar must be marked as read-only at the time they are loaded. If an attempt is made to load a file while it remains writable, the system throws an UnsatisfiedLinkError and execution is blocked. This reduces the attack surface where native code could be unauthorizedly tampered with right before execution, further improving app security.

[25] Officially, this is treated as an exception is thrown, but actual verification shows that an error occurs.
https://developer.android.com/about/versions/14/behavior-changes-14?hl=en#safer-dynamic-code-loading
[26] "Core app quality" Privacy and Security SC-E1
https://developer.android.com/docs/quality-guidelines/core-app-quality?hl=en

Fundamental Risks of DCL and Recommended Alternatives

To begin with, from a security perspective, using DCL itself is not recommended. Dynamically loading code carries risks of sensitive data theft or execution of harmful processes due to code injection or tampering. Users are unaware of what code the app is loading internally, and all responsibility for the app's behavior belongs to the app provider. In addition, DCL from remote sources can be a violation of Google Play policy.

Therefore, developers should first check whether the app or the SDKs being used employ System.load() or similar, and should avoid DCL unless there is a reasonable business justification. Officially, using Android App Bundle is recommended as an alternative, and it is desirable to transition to a design that includes features in the APK or App Bundle in advance as much as possible.

Security Countermeasures When Unavoidably Using DCL

If it is absolutely necessary due to app requirements to dynamically load native libraries or similar, take the following measures to mitigate tampering risks.

1. Use of access-restricted storage locations: As a storage location for dynamically loaded code, avoid locations that can be easily accessed from the outside (such as external storage), and use areas where access is strictly restricted, such as the app's internal storage or Scoped Storage.
2. Making files read-only: Ensure that the file is read-only before loading it.
3. Integrity checks and signature verification: Perform hash verification (such as SHA-256) of the file immediately before loading it to confirm that the file has not been tampered with. It is recommended to store the hash value itself in a secure area, and if possible, combine it with code signing and its verification to ensure the authenticity of the code.

4.6.3.8 Package Name of Media Owner

The MediaColumns class in MediaStore stores various types of information about files as columns, such as TITLE, DISPLAY_NAME, and MIME_TYPE. These column values can be used as query conditions to search for media files stored on the device.

Starting with Android 14, the column OWNER_PACKAGE_NAME, which represents the package name of the app that created the media file, has been removed and can no longer be specified as a query condition.

This change was made because information about installed apps is now considered part of a user's personal and confidential information.

The following is a sample code snippet that queries audio files stored on a device and attempts to display the OWNER_PACKAGE_NAME of each file.

```
cursor = contentResolver.query(
        MediaStore.Audio.Media.EXTERNAL_CONTENT_URI,
        null,null,null,null);

if (cursor != null && cursor.moveToFirst()) {

    Log.d(TAG, cursor.getString(cursor.getColumnIndex(MediaStore.Audio.Media.OWNER_
↪PACKAGE_NAME)));
```

The difference in behavior when this code is executed on Android 13 (API Level 33) and Android 14 (API Level 34) is shown below.

When executed on Android 13:

```
D  voicerecorder.audiorecorder.voice
```

When executed on Android 14:

```
E  Unknown message received from debugger! ''
```

As shown, in Android 13 the package name of the app that created the media file is output, while in Android 14 the query using OWNER_PACKAGE_NAME is not allowed, resulting in an error message being displayed in Logcat.

4.6.3.9 Measures for Preventing Path Traversal by Zip Files

Starting with Android 14, for Zip file entries containing "…" or starting with "/", a ZipException is thrown.

An example of this type of Zip file is shown below.

```
$ unzip -l test.zip
Archive:  test.zip
  Length      Date    Time    Name
---------  ---------- -----   ----
   104565  2023/02/28 14:21   ../mkcsv.txt
---------                     -------
   104565                     1 file
```

In the above file, the entry for test.zip is ". /mkcsv.txt". The sample code and execution result when accessing such a Zip file from an application targeting Android 14 are as follows.

```
filename = this.getFilesDir().getPath() + "/test.zip";

try {
    in = new ZipInputStream(new FileInputStream(filename));

    while ((zipEntry = in.getNextEntry()) != null) {
    // ...
} catch (IOException e) {
    e.printStackTrace();
}
```

```
W  java.util.zip.ZipException: Invalid zip entry path: ../mkcsv.txt
W         at com.android.internal.os.SafeZipPathValidatorCallback.
↪onZipEntryAccess(SafeZipPathValidatorCallback.java:61)
```

As you can see in the execution result log, a ZipException is thrown, which occurs when getNextEntry() is executed.

This is a measure to protect against a path traversal vulnerability that allows access to unintended paths from relative paths, and it is intended to encourage developers to take some kind of action.

To temporarily opt out of this specification, execute dalvik.system.ZipPathValidator.clearCallback() as follows.

```
filename = this.getFilesDir().getPath() + "/test.zip";

try {
    in = new ZipInputStream(new FileInputStream(filename));

    if (Build.VERSION.SDK_INT >= 34) {
        ZipPathValidator.clearCallback();
    }

    while ((zipEntry = in.getNextEntry()) != null) {
    // ...
} catch (IOException e) {
    e.printStackTrace();
}
```

4.6.3.10 Query the most recent user's choice for accessing selected photos

Starting from Android 15, apps can now highlight only the most recently selected photos and videos when partial media access is granted. This feature improves the user experience for apps that frequently request access to photos and videos. To take advantage of this feature, apps must enable the QUERY_ARG_LATEST_SELECTION_ONLY argument when sending queries to MediaStore through a ContentResolver.

Security Advantages

This feature provides several important advantages to user privacy and data security

1. limited access scope: The QUERY_ARG_LATEST_SELECTION_ONLY argument allows apps to access only the user's most recently selected photos and videos. This prevents the app from accessing the user's entire media library and limits unnecessary data access.
2. minimal data leakage risk: by granting partial access privileges, access is granted only to the minimum necessary data. This reduces the risk of apps unintentionally accessing other personal photos and videos.
3. User peace of mind: Users feel more secure knowing that the app will only access specific photos and videos, rather than all of their media files. This makes the app more trustworthy and increases user engagement.
4. Compliance: In recent years, many countries and regions have strengthened their data privacy laws and regulations, such as the General Data Protection Regulation (GDPR) and the California Consumer Privacy Act (CCPA). Using this feature makes it easier for apps to comply with these laws and regulations.
5. Selective Data Utilization: App developers can improve the user experience while protecting user data by allowing users to grant access only to specific photos and videos. This optimizes data usage and prevents unnecessary data collection.

The following is an example of a specific implementation

```
val externalContentUri = MediaStore.Files.getContentUri("external")

val mediaColumns = arrayOf(
    FileColumns._ID,
    FileColumns.DISPLAY_NAME,
    FileColumns.MIME_TYPE,
)

val queryArgs = bundleOf(
    // Return only items from the last selection (selected photos access)
    QUERY_ARG_LATEST_SELECTION_ONLY to true,
    // Sort returned items chronologically based on when they were added to the␣
↪device's storage
    QUERY_ARG_SQL_SORT_ORDER to "${FileColumns.DATE_ADDED} DESC",
    // Select media type as image or video
    QUERY_ARG_SQL_SELECTION to "${FileColumns.MEDIA_TYPE} = ? OR ${FileColumns.
↪MEDIA_TYPE} = ?",
    QUERY_ARG_SQL_SELECTION_ARGS to arrayOf(
        FileColumns.MEDIA_TYPE_IMAGE.toString(),
        FileColumns.MEDIA_TYPE_VIDEO.toString()
    )
)
```

This code allows the app to obtain access to the user's most recently selected photos and videos when partial media permissions are granted; by using the QUERY_ARG_LATEST_SELECTION_ONLY argument, the query results are limited to the most recently selected content, The QUERY_ARG_SQL_SORT_ORDER argument allows the results to be ordered chronologically. This allows users to efficiently access recently added media files.

4.6.3.11 MediaStore version lockdown

Starting with Android 16 (API Level 36), the specification for the value returned by MediaStore#getVersion() has changed significantly. Previously, this method returned a unique version string for each device, which included information such as a UUID that could identify the device or user.

This posed a significant privacy concern: apps could use this version value to identify devices and users. By sending the MediaStore#getVersion() value to a server, it was possible to track user behavior and fingerprint devices.

To mitigate this risk, starting with Android 16, the return value of MediaStore#getVersion() is unique to each app. Under the new specification, the returned value is a short hexadecimal string such as "149485ba50792615" and contains no information that can identify the device or user. Because different apps on the same device receive different values, comparing values across apps is meaningless.

This version value is intended to be used to detect whether the MediaStore state has changed significantly. For example, if an application caches MediaStore data, it can rebuild or resynchronize the cache when this version value changes. Processing that depends on the internal implementation, such as the content, format, or timing of the change, should be avoided; the correct usage is to assume that a change in the version value means that some major change has occurred.

```
val version = MediaStore.getVersion(this)
Log.d("MediaStoreVersion", "MediaStore#getVersion() = $version")
```

```
In case of Target SDK 35
2025-05-26 14:42:10.272 28868-28868 MediaStoreVersion       com.example.
↪myapplication            D  MediaStore#getVersion() = 1506:f4cbc590-2645-4417-
↪9177-99db10dcafbb

In case of Target SDK 36
2025-05-26 14:47:22.408 29089-29089 MediaStoreVersion       com.example.
↪myapplication            D  MediaStore#getVersion() = 149485ba50792615
```

4.6.3.12 App-owned photos

In Android 16 (API Level 36), when an app requests photo or video storage permission (e.g., READ_MEDIA_IMAGES) and the user grants "selected media only," photos and videos stored by the app via MediaStore (app-owned photos) are automatically preselected in the permission dialog. The user can disable this preselection, causing the app to lose access to those photos and videos. Preselection only occurs in the permission dialog and cannot be confirmed when using the standard Photo Picker API. This behavior does not occur in apps that only use the Photo Picker.

Developers must understand this specification and clearly indicate in their design, UI, and description that their app's photos will be preselected when requesting permission and that the user can disable this selection. If the user disables preselection, they will no longer be able to access those photos. Therefore, always check for permission when accessing images or videos, and if access is denied, implement appropriate exception handling and guidance, such as directing the user to the Settings app or offering to reselect. Google recommends using the Photo Picker, and generally recommends that apps do not declare storage permissions. This specification should only be taken into consideration when implementing an app if storage permission is required for special purposes, such as a gallery app.

In actual testing, by saving photos owned by the app via MediaStore and requesting storage permission, the app can confirm that its own photos are preselected in the permission dialog. If the user unchecks this box to allow permission, the app can also confirm that access to those photos will be lost. If storage permission is requested again, the dialog will not be displayed, and the permission status can only be changed from the Android Settings app.

Below is an example of the Compose sample code used to test this specification.

```
@Composable
fun AppOwnedPhotoPermissionTestScreen(modifier: Modifier = Modifier) {
    val context = LocalContext.current
    var savedUri by remember { mutableStateOf<android.net.Uri?>(null) }
```

(continues on next page)

(continued from previous page)

```
    var bitmap by remember { mutableStateOf<Bitmap?>(null) }
    var message by remember { mutableStateOf("") }
    var permissionGranted by remember { mutableStateOf(false) }

    val permissionLauncher = rememberLauncherForActivityResult(
        ActivityResultContracts.RequestPermission()
    ) { isGranted ->
        permissionGranted = isGranted
        message = if (isGranted) {
            " Permission granted (preselect behavior confirmed)"
        } else {
            " Permission denied"
        }
        if (isGranted && savedUri != null) {
            try {
                context.contentResolver.openInputStream(savedUri!!)?.use { input ->
                    bitmap = android.graphics.BitmapFactory.decodeStream(input)
                }
                message += "\n view saved photos"
            } catch (e: Exception) {
                // Implement proper exception handling and user guidance here.
                // Example: Displaying a dialog box in the Settings app to prompt␣
↪you to confirm or change permissions
                message += "\n failed to access photos. Please check your␣
↪permissions in settings."
            }
        }
    }

    Column(modifier = modifier.padding(16.dp)) {
        Button(onClick = {
            val resolver = context.contentResolver
            val imageCollection = MediaStore.Images.Media.getContentUri(MediaStore.
↪VOLUME_EXTERNAL_PRIMARY)
            val values = ContentValues().apply {
                put(MediaStore.Images.Media.DISPLAY_NAME, "test_app_owned_photo_$
↪{System.currentTimeMillis()}.jpg")
                put(MediaStore.Images.Media.MIME_TYPE, "image/jpeg")
            }
            val uri = resolver.insert(imageCollection, values)
            if (uri != null) {
                val bmp = Bitmap.createBitmap(100, 100, Bitmap.Config.ARGB_8888)
                resolver.openOutputStream(uri)?.use { out ->
                    bmp.compress(Bitmap.CompressFormat.JPEG, 100, out)
                }
                savedUri = uri
                message = "Save app-owned photos"
            } else {
                message = "Failed to save photo"
            }
        }) {
            Text("Save app-owned photos")
        }

        Spacer(Modifier.height(16.dp))
```

(continues on next page)

(continued from previous page)

```
        Button(onClick = {
            val permission = if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.
↪TIRAMISU) {
                Manifest.permission.READ_MEDIA_IMAGES
            } else {
                Manifest.permission.READ_EXTERNAL_STORAGE
            }
            permissionLauncher.launch(permission)
        }) {
            Text("Request storage permissions")
        }

        Spacer(Modifier.height(16.dp))

        if (bitmap != null) {
            Image(bitmap = bitmap!!.asImageBitmap(), contentDescription = null,␣
↪modifier = Modifier.size(100.dp))
        }

        Text(message, modifier = Modifier.padding(top = 16.dp))

        if (savedUri != null) {
            Text("Saved photos URI: $savedUri", style = MaterialTheme.typography.
↪bodySmall)
        }
    }
}
```

4.7 Using Intent

Intents are the fundamental mechanism for inter-component communication in Android, used to launch Activities, Services, or Broadcast Receivers within or outside the application. There are explicit Intents and implicit Intents, which must be chosen appropriately according to their purpose and security requirements.

Explicit Intents specify the target component class directly and are mainly used within the application. Implicit Intents, on the other hand, specify actions, data types, and similar parameters, allowing the system to select a matching component. This enables integration with other apps and invocation of system-standard features.

When using Intents, it is important to be aware of security risks such as unnecessary information leakage, excessive permission grants, misconfigured exported attributes, or incorrect Intent Filter settings. In particular, when using implicit Intents, there is a possibility that unintended apps could receive them, so thorough control of the information sent and validation upon receipt is required.

Among implicit Intents, those with the category android.intent.category.BROWSABLE are called Browsable Intents. Browsable Intents are mainly used to launch an Activity in response to deep links from a web browser or other apps (e.g., URLs using the https scheme). By specifying the BROWSABLE category in the intent-filter of the Manifest, it is possible to allow invocation from external sources.

Activities that accept Browsable Intents will receive input from external sources, making it essential to validate any data or parameters passed via the Intent to prevent malicious requests or data leaks. Avoid assigning the BROWSABLE category to Activities unnecessarily, and ensure proper implementation of parameter validation and access control as part of robust security measures.

With a solid understanding of the basics and precautions for Intents in general, when designing and implementing Browsable Intents, it is crucial to fully recognize the risks inherent to external integration points and take appropriate countermeasures.

4.7.1 Sample Code

Sample codes of an application which uses 'Browsable Intent' are shown below. Install 'Starter.html' on the web server and run it.

Points:

1. (Webpage side) Sensitive information must not be included.
2. Handle the URL parameter carefully and securely.

```
Starter.html
<html>
    <body>
        <!-- *** POINT 1 *** Sensitive information must not be included. -->
        <!-- Character strings to be passed as URL parameter, should be UTF-8 and
↪URI encoded. -->
        <a href="secure://jssec?user=user_id"> Login </a>
    </body>
</html>
```

```
AndroidManifest.xml
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
          >

  <application
      android:icon="@drawable/ic_launcher"
      android:label="@string/app_name"
      android:allowBackup="false" >
    <activity
        android:name=".BrowsableIntentActivity"
        android:label="@string/title_activity_browsable_intent"
        android:exported="true" >
      <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER" />
      </intent-filter>

      <intent-filter>
        <action android:name="android.intent.action.VIEW" />
        <!-- Accept implicit Intent -->
        <category android:name="android.intent.category.DEFAULT" />
        <!-- Accept Browsable intent -->
        <category android:name="android.intent.category.BROWSABLE" />
        <!-- Accept URI 'secure://jssec' -->
        <data android:scheme="secure" android:host="jssec"/>
      </intent-filter>
    </activity>
  </application>

</manifest>
```

```
BrowsableIntentActivity.java
/*
 * Copyright (C) 2012-2026 Japan Smartphone Security Association
 *
 * Licensed under the Apache License, Version 2.0 (the "License");
```

(continues on next page)

(continued from previous page)

```
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 *
 *      http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */

package org.jssec.android.browsableintent;

import android.app.Activity;
import android.content.Intent;
import android.net.Uri;
import android.os.Bundle;
import android.widget.TextView;

public class BrowsableIntentActivity extends Activity {

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_browsable_intent);

        Intent intent = getIntent();
        Uri uri = intent.getData();
        if (uri != null) {
            // Get UserID which is passed by URI parameter
            // *** POINT 2 *** Handle the URL parameter carefully and securely.
            // Omitted, since this is a sample. Please refer to
            // "3.2 Handling Input Data Carefully and Securely."
            String userID = "User ID = " +  uri.getQueryParameter("user");
            TextView tv = (TextView)findViewById(R.id.text_userid);
            tv.setText(userID);
        }
    }

}
```

4.7.2 Rule Book

Follow rules listed below when using "Browsable Intent".

1. *(Webpage side) Sensitive Information Must Not Be Included in Parameter of Corresponding Link (Required)*
2. *Handle the URL Parameter Carefully and Securely (Required)*
3. *If possible, use Android App Links (Recommended)*

4.7.2.1 (Webpage side) Sensitive Information Must Not Be Included in Parameter of Corresponding Link (Required)

When tapping the link in browser, an intent which has a URL value in its data (It can be retrieve by Intent#getData) is issued, and an application which has a corresponding Intent Filter is launched from Android system.

At this moment, when there are several applications which Intent Filter is set to receive the same URI scheme, application selection dialogue is shown in the same way as normal launch by implicit Intent, and an application which user selected is launched. In case that a Malware is listed in the selection of application selection dialogue, there is a risk that user may launch the Malware by mistake and parameters in URL are sent to Malware.

As per above, it is necessary to avoid from include sensitive information directly in URL parameter as it is for creating general Webpage link since all parameters which are included in Webpage link URL can be given to Malware.

Example that User ID and Password are included in URL.

insecure://sample/login?userID=12345&password=abcdef

In addition, there is a risk that user may launch a Malware and input password to it when it is defined in specs that password input is executed in an application after being launched by 'Browsable Intent', even if the URL parameter includes only non-sensitive information like User ID. So it should be considered that specs like a whole Login process is completed within application side. It must be kept in mind when designing an application and a service that launching application by 'Browsable Intent' is equivalent to launching by implicit Intent and there is no guarantee that a valid application is launched.

4.7.2.2 Handle the URL Parameter Carefully and Securely (Required)

URL parameters which are sent to an application are not always from a legitimate Web page, since a link which is matched with URI scheme can be made by not only developers but anyone. In addition, there is no method to verify whether the URL parameter is sent from a valid Web page or not.

So it is necessary to verify safety of a URL parameter before using it, e.g. check if an unexpected value is included or not.

4.7.2.3 If possible, use Android App Links (Recommended)

Android provides a mechanism called App Links, which allows linking a website URL with an app. By associating URLs with an app, even if multiple apps on the device specify the same URL, the system can identify the legitimate app associated with the URL. Therefore, even if a malware app specifying the same link is installed on the device, the malware app will not be launched from the link, and the legitimate app will start without requiring user selection. This feature is available on devices running Android 6.0 (API Level 23) and later.

To use App Links, ownership of both the app and the website must be verified. Developers need to prepare a JSON file describing the app's package name and certificate fingerprint, and publish it on their website. Then, by specifying the same website link in the Browsable Intent's intent filter and declaring android:autoVerify="true", the system verifies the link and enables its use.

```
https://www.jssec.org/.well-known/assetlinks.json
[{
    "relation": ["delegate_permission/common.handle_all_urls"],
    "target": {
        "namespace": "android_app",
        "package_name": "",
        "sha256_cert_fingerprints":
        [""]
    }
}]
```

```
AndroidManifest.xml
    <activity
        android:name=".BrowsableIntentActivity"
        android:exported="true" >
      <!-- android:autoVerify="true" -->
      <intent-filter android:autoVerify="true">
        <action android:name="android.intent.action.VIEW" />
        <category android:name="android.intent.category.DEFAULT" />
        <category android:name="android.intent.category.BROWSABLE" />
        <!-- URI 'https://jssec.org'  -->
        <data android:scheme="https" android:host="jssec.org" />
      </intent-filter>
    </activity>
```

4.7.3 Advanced

4.7.3.1 Component Export Control and Intent Sending Restrictions

Starting with Android 12 (API Level 31), components that define an intent-filter (Activity, Service, Broadcast Receiver) cannot be built unless the android:exported attribute is explicitly set. If the exported attribute is omitted, a warning will be displayed when editing the manifest, and an error will occur at build time. The previous behavior, in which the presence or absence of an intent-filter implicitly determined whether a component was public or private, has been abolished in favor of a specification designed to prevent security risks caused by design errors. If exported=”true”, the component can be called externally, and if exported=”false”, it cannot be called externally. Which attribute to specify should be determined based on the design intent. This applies to all Activities, Services, and Broadcast Receivers. When using an intent-filter, be sure to explicitly specify the exported attribute.

```
<activity
    android:name=".SampleActivity"
    android:exported="false">
    <intent-filter>
        <action android:name="com.example.ACTION_SAMPLE" />
        <category android:name="android.intent.category.DEFAULT" />
    </intent-filter>
</activity>
```

Starting with Android 14, an exception occurs when an implicit intent is sent to an internal component with exported=”false”. For example, sending an implicit intent to a private activity would launch the activity up until Android 13, but starting with Android 14, an ActivityNotFoundException occurs. This is to prevent the risk of launching an unintended activity if the intent filter definition conflicts with that of another app. For private components, use explicit intents using setPackage or setClassName. The exported attribute, intent filter, and intent type (explicit/implicit) must be considered as a single design.

```
Intent intent = new Intent("com.example.myapplication.MyAction");
intent.setPackage(getApplicationContext().getPackageName());
startActivity(intent);
```

4.7.3.2 PendingIntent Mutability

PendingIntent is a mechanism that allows an app to delegate an “intent to be executed later” to another app or system using its own permissions. When creating an intent, you specify the intent and various flags, but specifying “mutability” is particularly important from a security perspective. A mutable PendingIntent allows the destination app or system to change parts of the intent. Because unset fields can be overwritten using methods such as fillIn(), there is a risk that a malicious app could cause the intent to behave in an unexpected way.

Apps targeting Android 12 or later must specify the mutability of a PendingIntent object. Mutable is specified with the PendingIntent.FLAG_MUTABLE flag, and immutable is specified with the PendingIntent.FLAG_IMMUTABLE flag. Attempting to create a PendingIntent object without specifying either flag will result in an IllegalArgumentException. While specifying FLAG_IMMUTABLE is recommended for enhanced security, some features, such as direct reply actions, require FLAG_MUTABLE.

```
PendingIntent pendingIntent = PendingIntent.getActivity(getApplicationContext(),␣
↪REQUEST_CODE, intent, PendingIntent.FLAG_IMMUTABLE);
// When mutability is required, for example in direct reply actions
PendingIntent pendingIntent = PendingIntent.getActivity(getApplicationContext(),␣
↪REQUEST_CODE, intent, PendingIntent.FLAG_MUTABLE);
```

4.7.3.3 Strict Enforcement of Intent Filter

Starting with Android 13 (API Level 33), when launching an activity with an explicit intent, the intent is blocked if it does not match any of the receiving activity's intent filters. However, this restriction does not apply to components that do not have an intent filter defined, intents sent from within the same application, or intents sent from the system or root.

As a concrete example, we created a public activity with the intent filter defined below. To confirm that unmatched intents are blocked, we used the adb command to launch the public activity using an explicit intent with a different action. Here, we confirmed that the activity could not be launched when using the not_match_intent_filter, which is not defined in the intent filter.

AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    >

    <application
        android:allowBackup="false"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/Theme.PublicActivity_33">
        <!-- Public activity -->
        <!--intent-filter is defined-->
        <activity
            android:name=".IntentFilterTestActivity"
            android:exported="true">
            <intent-filter>
                <action android:name="api33_intent_block_test1" />
                <action android:name="api33_intent_block_test2" />
            </intent-filter>
        </activity>

        <activity
            android:name=".MainActivity"
            android:exported="true">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>
```

(continues on next page)

(continued from previous page)

```
</manifest>
```

```
Verifying start of an activity by adb command

>adb shell am start -n org.jssec.android.activity.publicactivity_33/.
↪IntentFilterTestActivity
Starting: Intent { cmp=org.jssec.android.activity.publicactivity_33/.
↪IntentFilterTestActivity }


>adb shell am start -a api33_intent_block_test1 -n org.jssec.android.activity.
↪publicactivity_33/.IntentFilterTestActivity
Starting: Intent { act=api33_intent_block_test1 cmp=org.jssec.android.activity.
↪publicactivity_33/.IntentFilterTestActivity }


>adb shell am start -a not_match_intent_filter -n org.jssec.android.activity.
↪publicactivity_33/.IntentFilterTestActivity
Starting: Intent { act=not_match_intent_filter cmp=org.jssec.android.activity.
↪publicactivity_33/.IntentFilterTestActivity }
Error type 3
Error: Activity class {org.jssec.android.activity.publicactivity_33/org.jssec.
↪android.activity.publicactivity_33.IntentFilterTestActivity} does not exist.


>adb shell am start -a api33_intent_block_test2 -n org.jssec.android.activity.
↪publicactivity_33/.IntentFilterTestActivity
Starting: Intent { act=api33_intent_block_test2 cmp=org.jssec.android.activity.
↪publicactivity_33/.IntentFilterTestActivity }
```

Additionally, starting with Android 15, new security measures have been introduced to make intents safer and more robust, such as exact match of intent filters and mandatory actions. Intents that do not match the target intent filter will not be received, and intents without an action will not match the intent filter. Similar restrictions apply to pending intents. Intents should be designed to match all intent filter actions, data, and categories, and an action must be set for the intent.

```
<activity
    android:name="com.example.targetapp.TargetActivity">
    <intent-filter>
        <action android:name="android.intent.action.VIEW" />
        <category android:name="android.intent.category.DEFAULT" />
        <data android:scheme="content" />
    </intent-filter>
</activity>
```

```
val intent = Intent().apply {
    action = Intent.ACTION_VIEW
    data = Uri.parse("content://com.example.provider/item")
    setClassName("com.example.targetapp", "com.example.targetapp.TargetActivity")
}
startActivity(intent)
```

Design intent-filters for exported Activities to be minimal and avoid creating unnecessary entry points. Enable StrictMode's detectUnsafeIntentLaunch() to verify that there are no matching violations or Intents sent without specifying an action.

```
StrictMode.setVmPolicy(
    new StrictMode.VmPolicy.Builder()
        .detectUnsafeIntentLaunch()
        .penaltyLog()
        .build()
);
```

4.7.3.4 Expanded IntentFilter capabilities

Android 15 introduced UriRelativeFilter and UriRelativeFilterGroup, enabling detailed matching including query parameters and fragments in addition to the traditional scheme, host, and path. This allows for flexible reception control, such as accepting only requests with specific query parameters for specific purposes, such as authentication flows and campaigns. Filter conditions should be kept to a minimum, and unnecessary requests should not be accepted. Verify reception behavior using adb, etc., and test to ensure that the app is not launched with an unintended URI.

```
val filterGroup = UriRelativeFilterGroup(UriRelativeFilterGroup.ACTION_ALLOW).
↪apply {
    addUriRelativeFilter(UriRelativeFilter(UriRelativeFilter.PATH, PatternMatcher.
↪PATTERN_PREFIX, "/auth"))
    addUriRelativeFilter(UriRelativeFilter(UriRelativeFilter.QUERY, PatternMatcher.
↪PATTERN_LITERAL, "token=securetoken"))
}
val isMatch = uri?.let { filterGroup.matchData(it) } ?: false
```

```
When the URI matches the filter:
$ adb shell am start -W -a android.intent.action.VIEW -d "https://example.com/auth?
↪token=securetoken"

When the URI does not match the filter:
$ adb shell am start -W -a android.intent.action.VIEW -d "https://example.com/auth?
↪token=invalidtoken"
```

4.7.3.5 Changes to PendingIntent and Package Stopped State

The FLAG_STOPPED state of a package (entered when a user long-presses the app icon and selects “Force stop”) is intended to keep the app in this state until the user explicitly removes it by directly launching the app or indirectly interacting with it, such as via a sharesheet or widget.

In previous Android versions, even if the app was in the stopped state, any pending PendingIntents remained valid, and the system did not cancel them. This allowed the app to become active again unintentionally.

In Android 15, the system behavior was updated to align with the intended semantics of FLAG_STOPPED. Now, the app can only be removed from the stopped state through direct or indirect user actions.

This change may affect existing apps in the following ways:

1. When the app enters the stopped state, all PendingIntents are canceled, which may cause important notifications or activities to be lost.
2. Apps using widgets will see their widgets grayed out and become unresponsive to user interaction.

Existing apps need to add logic to re-register PendingIntents when the app restarts.

```
override fun onReceive(context: Context, intent: Intent) {
    if (Intent.ACTION_BOOT_COMPLETED == intent.action) {
        // Handle the app restart after boot completed
        reinitializePendingIntents(context)
```

(continues on next page)

(continued from previous page)

```
    }
}

private fun reinitializePendingIntents(context: Context) {
    // Logic to re-register pending intents
    val intent = Intent(context, MyReceiver::class.java)
    val pendingIntent = PendingIntent.getBroadcast(context, 0, intent,␣
↪PendingIntent.FLAG_UPDATE_CURRENT)
    // Re-schedule using AlarmManager if needed
    val alarmManager = context.getSystemService(Context.ALARM_SERVICE) as␣
↪AlarmManager
    alarmManager.setExact(AlarmManager.RTC_WAKEUP, System.currentTimeMillis() +␣
↪60000, pendingIntent)
}
```

You can check the FLAG_STOPPED state using the ApplicationStartInfo.wasForceStopped() method.

```
if (ApplicationStartInfo.wasForceStopped()) {
    // Handle the case where the app was force stopped
    reinitializePendingIntents()
}
```

4.7.3.6 Safer Intents

Safer Intents is an intent security enhancement feature introduced in Android 16. Enabling android:intentMatchingFlags="enforceIntentFilter" in the manifest prevents the reception of intents that do not match the intent filter, even if the intent is an explicit intent (component name specified). Previously, explicit intents could be received even if the action was not specified, but this specification reduces the risk of receiving unintended intents and information leakage. This feature only applies to communication between multiple apps and does not affect intent transmission within the same app.

```
<application
    android:intentMatchingFlags="enforceIntentFilter"
    ...>
    <receiver
        android:name=".MyBroadcastReceiver"
        android:exported="true">
        <intent-filter>
            <action android:name="com.example.MY_CUSTOM_ACTION" />
        </intent-filter>
    </receiver>
</application>
```

```
val intent = Intent()
intent.component = ComponentName("com.example.myapplication", "com.example.
↪myapplication.MyBroadcastReceiver")
// with action
intent.action = "com.example.MY_CUSTOM_ACTION"
sendBroadcast(intent)
// no action
val intentNoAction = Intent()
intentNoAction.component = ComponentName("com.example.myapplication", "com.example.
↪myapplication.MyBroadcastReceiver")
sendBroadcast(intentNoAction)
```

When this code is executed, only explicit intents with an action will be received, and intents without an action will not be received. Enabling Safer Intents will reliably block unintended intent reception.

This mechanism forms part of the roadmap to enhance intent communication security on the Android platform in future versions. It is likely to be enabled by default in the future, and even now Google recommends that both new and existing apps actively adopt it.

4.7.3.7 Countermeasures against Intent redirect attacks

By explicitly setting the exported attribute of an Activity to "false," you can prevent direct launch from other apps. However, in environments up to Android 15 (API Level 35), there was an issue where if a private Activity was passed as a sublevel Intent via an Intent's extra from another app, it could be unintentionally launched using startActivity. This was because, even though the private setting via the exported attribute was effective for "direct launch," the system did not provide sufficient protection against launches via redirects via Intent's extra, etc.

To address this issue, system-level defenses were introduced in Android 16 (API Level 36). If an Intent received from another app contains an Intent in its extras (sublevel Intent), attempting to launch it directly using startActivity or similar will now result in a security exception being thrown by default, blocking the launch. This essentially eliminates the risk of unintentionally launching Activities with exported="false" or components that should not be exposed externally via extras. If you must remove this protection, the only way to launch them is to explicitly call removeLaunchSecurityProtection(), but this API is only available on Android 16 and later.

In the actual testing, we prepared two patterns: an Intent from the sender app pointing to the receiver app's private Activity (exported="false") was stored in extra("sub_intent"), and this was then used as startActivity in the receiver app's MainActivity, or removeLaunchSecurityProtection() was called before startActivity. An example of the testing code is shown below.

senderapp side (sender side of the attack scenario)

```
val subIntent = Intent().apply {
    setClassName("com.example.receiverapp", "com.example.receiverapp.
↪PrivateActivity")
}
val topIntent = Intent().apply {
    setClassName("com.example.receiverapp", "com.example.receiverapp.MainActivity")
    putExtra("sub_intent", subIntent)
    addFlags(Intent.FLAG_ACTIVITY_NEW_TASK)
}
startActivity(topIntent)
```

receiverapp side (receiving/launching side)

```
val iSublevel = intent.getParcelableExtra("sub_intent", Intent::class.java)
try {
    // In Android 16, a security exception occurs here, PrivateActivity cannot be␣
↪launched
    iSublevel?.let { startActivity(it) }
} catch (e: Exception) {
    // Display exception details in logs and UI
}

try {
    // Only valid for Android 16 and above. You can remove the protection by␣
↪calling removeLaunchSecurityProtection().
    iSublevel?.removeLaunchSecurityProtection()
    iSublevel?.let { startActivity(it) }
} catch (e: Exception) {
    // Display exception details in logs and UI
}
```

As a result of this verification, it was confirmed that in Android 15, PrivateActivity can be launched by simply calling startActivity, whereas in Android 16, it is blocked by a security exception unless removeLaunchSecurityProtection() is called. Furthermore, calling removeLaunchSecurityProtection() in an Android 15 environment results in a NoSuchMethodError and the app crashes, making it necessary to implement branching based on API level.

Table 4.7.1: Behavior of Intent Redirection Attack Countermeasures in Android 15/16

Procedure	Android 15API Level 35	Android 16API Level 36
startActivity as is	Starts successfully	Blocked by a security exception
removeLaunchSecurityProtection	Crash with NoSuchMethodError (API not implemented)	The protection is removed, and PrivateActivity is launched.

Developers should design and implement their apps with this security enhancement in mind for Android 16 and later. Developers should not simply launch intents included in the extras of intents received from other apps, but should thoroughly verify and sanitize the destination and content. For private activities, developers should not only explicitly mark exported=”false” but also be careful about redirects via extras, and design their apps to avoid unnecessary external exposure or unintended launches. Use of removeLaunchSecurityProtection() should be kept to a minimum, and if it is unavoidable, it is recommended that developers conduct thorough risk assessments and testing.

4.7.3.8 Restrictions on Implicit URI Grants

In Android 18 and later, implicit URI grants when sharing URIs with other apps are scheduled to be restricted. This change aims to prevent unintended file access in inter-app integration and strengthen security.

Conventionally, when sharing a file (URI) between apps, such as by sending an intent containing a URI like Intent.ACTION_SEND, the system automatically (implicitly) granted read (READ) or write (WRITE) permissions to the receiving app. However, under the restrictions in Android 18 and later, this implicit permission granting will no longer be performed. Therefore, if sent while omitting permission granting flags as before, the receiving side app will not be able to access the file, resulting in errors such as SecurityException or functions failing to operate correctly. When performing implicit URI permission granting in an Android 17 environment, a system error log such as Implicit URI grant for android.intent.action.SEND action will be discontinued from Android 18 onwards. Please set the grant explicitly in the app. is output to Logcat, prompting developers to take action.

As a solution, it is necessary to explicitly set access permission flags (Intent.FLAG_GRANT_READ_URI_PERMISSION or Intent.FLAG_GRANT_WRITE_URI_PERMISSION) in the sending app. Also, to delegate URI permissions to the receiving app more reliably, setting the URI not only in Intent.EXTRA_STREAM but also in clipData is recommended.

In particular, for apps with features such as the following, checking whether explicit permission granting is performed and taking action is strongly recommended.

- File sharing features for photos, PDFs, etc.
- File transmission to other apps / email attachment creation
- Sharing features to SNS, etc.
- Camera launch (passing image storage target URI via ACTION_IMAGE_CAPTURE)

The following shows a non-recommended implementation and an Android 18-compliant implementation example that explicitly grants permissions.

```
// NG Example: Traditional non-recommended implementation (relying on implicit URI␣
↪grants)
fun shareFileLegacy(context: Context, fileUri: Uri) {
    val intent = Intent(Intent.ACTION_SEND).apply {
        putExtra(Intent.EXTRA_STREAM, fileUri)
        type = "image/jpeg"
        // Conventionally, flags were often omitted because the system␣
↪automatically granted permissions
```

(continues on next page)

(continued from previous page)

```
    }
    val chooser = Intent.createChooser(intent, "Share without flags")
    context.startActivity(chooser)
}

// OK Example: Android 18-compliant implementation (explicit permission granting␣
↪and use of ClipData)
fun shareFileSafely(context: Context, fileUri: Uri) {
    val intent = Intent(Intent.ACTION_SEND).apply {
        putExtra(Intent.EXTRA_STREAM, fileUri)
        type = "image/jpeg"

        // (1) Explicitly grant READ permission flag to avoid restrictions in␣
↪Android 18 and later
        addFlags(Intent.FLAG_GRANT_READ_URI_PERMISSION)

        // (2) Set URI in ClipData as well to delegate permissions more reliably␣
↪(Recommended implementation)
        clipData = ClipData.newRawUri("", fileUri)
    }
    val chooser = Intent.createChooser(intent, "Share with permission granted")
    context.startActivity(chooser)
}
```

Note that when launching a camera app to take a photo and saving it to a URI prepared by your own app (ACTION_IMAGE_CAPTURE), file write permission must be granted to the camera app, so launch by explicitly specifying intent.addFlags(Intent.FLAG_GRANT_WRITE_URI_PERMISSION).

4.7.3.9 Cross-Profile Loopback Restriction

In Android 17 and later, loopback traffic (localhost / 127.0.0.1) between profiles, such as between personal profiles and work profiles, is restricted by default.

A profile is a mechanism for separating and managing apps and data for each user or purpose within an Android device, providing a feature that allows a single smartphone to have multiple independent usage environments. Typical examples include personal profiles used freely by individuals and Work Profiles managed by enterprises, with each profile treated as a separate area. Even for the same app, it operates as a separate installation for each profile, and stored data, settings, and accessible information are also separated, reducing risks of information leakage and unauthorized access.

Background of Security Enhancement

In previous versions, apps in different profiles on the same device could communicate via localhost in some cases, creating potential pathways for unintended data sharing and information leakage. In particular, a problem was identified where an app operating in one profile could listen as a local server and an app in another profile could connect to it, allowing data areas that should originally be separated to be crossed.

Due to the restriction on cross-profile loopback communication in Android 17, profiles are strictly separated as distinct users even within the same device. The risk of corporate business data stored in work profiles leaking to apps on the personal profile side, as well as attack techniques where malware acquires information via another profile, are significantly restricted.

Impact on Developers and Recommended Alternatives

Caution is required if cross-profile communication was previously implemented using local HTTP servers or localhost-based integration, as those implementations may stop functioning (loopback communication within the same profile is unaffected).

As an alternative when integration across profiles is necessary, migrating to a design that explicitly and securely performs cross-profile integration using standard APIs provided by Android, such as CrossProfileApps API, permission-

controlled Intent, or ContentProvider, is required. This change is an important specification update prioritizing security and isolation over convenience, and is a key point to note particularly in app development intended for enterprise use or high-security environments.

4.8 Outputting Log to LogCat

There's a logging mechanism called LogCat in Android, and not only system log information but also application log information are also output to LogCat. Log information in LogCat can be read out from other application in the same device[27], so the application which outputs sensitive information to Logcat, is considered that it has the vulnerability of the information leakage. The sensitive information should not be output to LogCat.

From a security point of view, in release version application, it's preferable that any log should not be output. However, even in case of release version application, log is output for some reasons in some cases. In this chapter, we introduce some ways to output messages to LogCat in a safe manner even in a release version application. Along with this explanation, please refer to "4.8.3.1. *Two Ways of Thinking for the Log Outputting in Release version application*".

4.8.1 Sample Code

Herein after, the method to control the Log output to LogCat by ProGuard in release version application. ProGuard is one of the optimization tools which automatically delete the unnecessary code like unused methods, etc.

There are five types of log output methods, Log.e(), Log.w(), Log.i(), Log.d(), Log.v(), in android.util.Log class. Regarding log information, intentionally output log information (hereinafter referred to as the Operation log information) should be distinguished from logging which is inappropriate for a release version application such as debug log (hereinafter referred to as the Development log information). It's recommended to use Log.e()/w()/i() for outputting operation log information, and to use Log.d()/v() for outputting development log. Refer to "4.8.3.2. *Selection Standards of Log Level and Log Output Method*" for the details of proper usage of five types of log output methods, in addition, also refer to "4.8.3.3. *DEBUG Log and VERBOSE Log Are Not Always Deleted Automatically*".

here's an example of how to use LogCat in a safe manner. This example includes Log.d() and Log.v() for outputting debug log. If the application is for release, these two methods would be deleted automatically. In this sample code, ProGuard is used to automatically delete code blocks where Log.d()/v() is called.

Points:

1. Sensitive information must not be output by Log.e()/w()/i(), System.out/err.
2. Sensitive information should be output by Log.d()/v() in case of need.
3. The return value of Log.d()/v() should not be used (with the purpose of substitution or comparison).
4. When you build an application for release, you should bring the mechanism that automatically deletes inappropriate logging method like Log.d() or Log.v() in your code.
5. An APK file for the (public) release must be created in release build configurations.

ProGuardActivity.java

```
/*
 * Copyright (C) 2012-2026 Japan Smartphone Security Association
 *
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 *
 *      http://www.apache.org/licenses/LICENSE-2.0
 *
```

(continues on next page)

[27] The log information output to LogCat can be read by applications that declare using READ_LOGS permission. However, in Android 4.1 and later, log information that is output by other application cannot be read. But smartphone user can read every log information output to logcat through ADB.

(continued from previous page)

```
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */

package org.jssec.android.log.proguard;

import android.app.Activity;
import android.os.Bundle;
import android.util.Log;

public class ProGuardActivity extends Activity {

    final static String LOG_TAG = "ProGuardActivity";

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_proguard);

        // *** POINT 1 *** Sensitive information must not be output by
        // Log.e()/w()/i(), System.out/err.
        Log.e(LOG_TAG, "Not sensitive information (ERROR)");
        Log.w(LOG_TAG, "Not sensitive information (WARN)");
        Log.i(LOG_TAG, "Not sensitive information (INFO)");

        // *** POINT 2 *** Sensitive information should be output by
        // Log.d()/v() in case of need.
        // *** POINT 3 *** The return value of Log.d()/v()should not be used
        // (with the purpose of substitution or comparison).
        Log.d(LOG_TAG, "sensitive information (DEBUG)");
        Log.v(LOG_TAG, "sensitive information (VERBOSE)");
    }
}
```

```
proguard-project.txt
# prevent from changing class name and method name etc.
-dontobfuscate

# *** POINT 4 *** In release build, the build configurations in which Log.d()/v()
# are deleted automatically should be constructed.
-assumenosideeffects class android.util.Log {
    public static int d(...);
    public static int v(...);
}
```

*** Point 5 *** An APK file for the (public) release must be created in release build configurations.

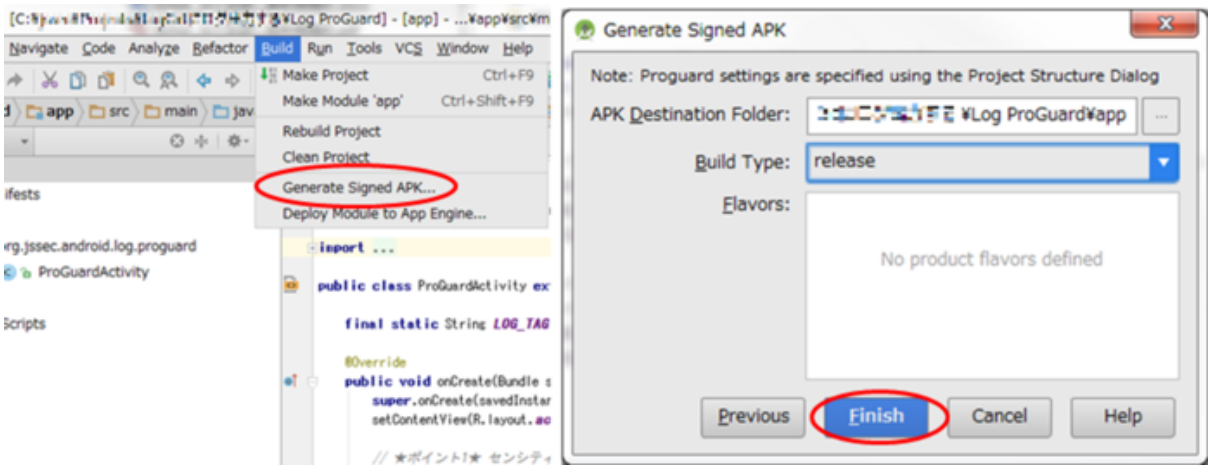


Fig. 4.8.1: How to create release version application

The difference of LogCat output between development version application (debug build) and release version application (release build) are shown in below Fig. 4.8.2

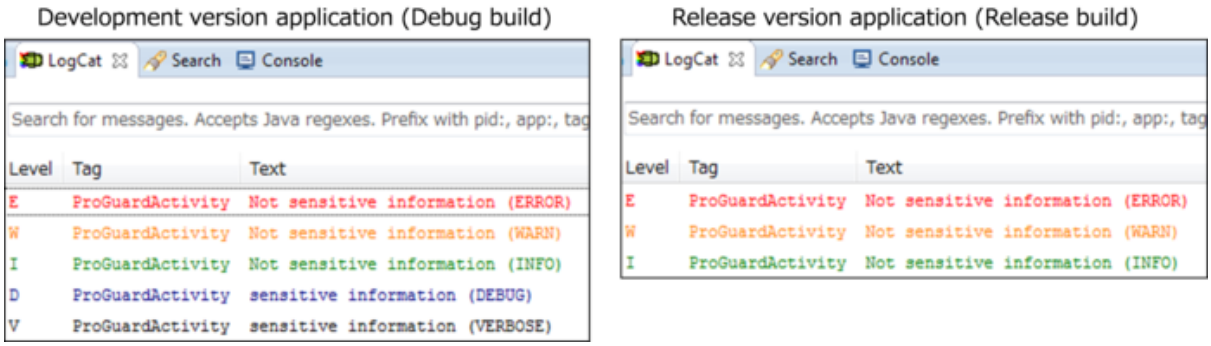


Fig. 4.8.2: Difference of LogCat output between development version application and release version application

4.8.2 Rule Book

When you output log messages, follow the rules below.

1. *Sensitive Information Must Not Be Included in Operation Log Information (Required)*
2. *Construct the Build System to Auto-delete Codes which Output Development Log Information When Build for the Release (Recommended)*
3. *Use Log.d()/v() Method When Outputting Throwable Object (Recommended)*
4. *Use Only Methods of the android.util.Log Class for the Log Output (Recommended)*

4.8.2.1 Sensitive Information Must Not Be Included in Operation Log Information (Required)

Log which was output to LogCat can be read out from other applications, so sensitive information like user's login information should not be output by release version application. It's necessary not to write code which outputs sensitive information to log during development, or it's necessary to delete all of such codes before release.

To follow this rule, first, not to include sensitive information in operation log information. In addition, it's recommended to construct the system to delete code which outputs sensitive information when build for release. Please refer to "4.8.2.2. *Construct the Build System to Auto-delete Codes which Output Development Log Information When Build for the Release (Recommended)*".

4.8.2.2 Construct the Build System to Auto-delete Codes which Output Development Log Information When Build for the Release (Recommended)

When application development, sometimes it's preferable if sensitive information is output to log for checking the process contents and for debugging, for example the interim operation result in the process of complicated logic, information of program's internal state, communication data structure of communication protocol. It doesn't matter to output the sensitive information as debug log during developing, in this case, the corresponding log output code should be deleted before release, as mentioned in "4.8.2.1. *Sensitive Information Must Not Be Included in Operation Log Information (Required)*".

To delete surely the code which outputs development log information when release builds, the system which executes code deletion automatically by using some tools, should be constructed. ProGuard, which was described in "4.8.1. *Sample Code*", can work for this method. As described below, there are some noteworthy points on deleting code by ProGuard. Here it's supposed to apply the system to applications which output development log information by either of Log.d()/v(), based on "4.8.3.2. *Selection Standards of Log Level and Log Output Method*".

ProGuard deletes unnecessary code like unused methods, automatically. By specifying Log.d()/v() as parameter of -assumenosideeffects option, call for Log.d(), Log.v() are granted as unnecessary code, and those are to be deleted.

By specifying -assumenosideeffects to Log.d()/v(), make it auto-deletion target.

```
-assumenosideeffects class android.util.Log {
    public static int d(...);
    public static int v(...);
}
```

In case using this auto deletion system, pay attention that Log.v()/d() code is not deleted when using returned value of Log.v(), Log.d(), so returned value of Log.v(), Log.d(), should not be used. For example, Log.v() is not deleted in the next examination code.

Examination code which Log.v() that is specifeied to be deleted is not deketed.

```
int i = android.util.Log.v("tag", "message");
//Use the returned value of Log.v() for examination.
System.out.println(String.format("Log.v() returned %d.", i));
```

If you'd like to reuse source code, you should keep the consistency of the project environment including ProGuard settings. For example, source code that presupposes Log.d() and Log.v() are deleted automatically by above ProGuard setting. If using this source code in another project which ProGuard is not set, Log.d() and Log.v() are not to be deleted, so there's a risk that the sensitive information may be leaked. When reusing source code, the consistency of project environment including ProGuard setting should be secured.

4.8.2.3 Use Log.d()/v() Method When Outputting Throwable Object (Recommended)

As mentioned in "4.8.1. *Sample Code*" and "4.8.3.2. *Selection Standards of Log Level and Log Output Method*", sensitive information should not be output to log through Log.e()/w()/i(). On the other hand, in order that a developer wants to output the details of program abnormality to log, when exception occurs, stack trace is output to LogCat by Log.e(..., Throwable tr)/w(..., Throwable tr)/i(..., Throwable tr), in some cases. However, sensitive information may sometimes be included in the stack trace because it shows detail internal structure of the program. For example, when SQLiteException is output as it is, what type of SQL statement is issued is clarified, so it may give the clue for SQL injection attack. Therefore, it's recommended that use only Log.d()/Log.v() methods, when outputting throwable object.

4.8.2.4 Use Only Methods of the android.util.Log Class for the Log Output (Recommended)

You may output log by System.out/err to verify the application's behavior whether it works as expected or not, during development. Of course, log can be output to LogCat by print()/println() method of System.out/err, but it's strongly recommended to use only methods of android.util.Log class, by the following reasons.

When outputting log, generally, use the most appropriate output method properly based on the urgency of the information, and control the output. For example, categories like serious error, caution, simple application's information notice, etc. are to be used. However, in this case, information which needs to be output at the time of release (operation log information) and information which may include the sensitive information (development log information) are output by the same method. So, it may happen that when delete code which outputs sensitive information, it's in danger that some deletion are dropped by oversight.

Along with this, when using android.util.Log and System.out/err for log output, compared with using only android.util.Log, what needs to be considered will increase, so it's in danger that some mistakes may occur, like some deletion are dropped by oversight.

To decrease risk of above mentioned mistakes occurrence, it's recommended to use only methods of android.util.Log class.

4.8.3 Advanced Topics

4.8.3.1 Two Ways of Thinking for the Log Outputting in Release version application

There are two ways of thinking for log output in release version application. One is any log should never be output, and another is necessary information for later analysis should be output as log. It's favorable that any log should never be output in release version application from the security point of view, but sometimes, log is output even in release version application for various reasons. Each way of thinking is described as per below.

The former is "Any log should never be output", this is because outputting log in release version application is not so much valuable, and there is a risk to leak sensitive information. This comes from there's no method for developers to collect log information of the release version application in Android application operation environment, which is different from many Web application operation environments. Based on this thinking, the logging codes are used only in development phase, and all the logging codes are deleted on building release version application.

The latter is "necessary information should be output as log for the later analysis", as a final option to analyze application bugs in customer support, in case of any questions or doubt to your customer support. Based on this idea, as introduced above, it is necessary to prepare the system that prevent human errors and bring it in your project because if you don't have the system you have to keep in mind to avoid logging the sensitive information in release version application.

For more details about logging method, refer to the following document.

Code Style Guidebook for Contributors / Log Sparingly

> https://source.android.com/setup/contribute/code-style#log-sparingly

4.8.3.2 Selection Standards of Log Level and Log Output Method

There are five levels of log level (ERROR, WARN, INFO, DEBUG, VERBOSE) are defined in android.util.Log class in Android. You should select the most appropriate method when using the android.util.Log class to output log messages according to Table 4.8.1 which shows the selection standards of logging levels and methods.

Table 4.8.1: Selection standards of log levels and log output method

Log level	Method	Log information to be output	Cautions for application release
ERROR	Log.e()	Log information which is output when application is in a fatal state.	Log information as per left may be referred by users, so it could be output both in development version application and in release version application. Therefore, sensitive information should not be output in these levels.
WARN	Log.w()	Log information which is output when application faces the unexpected serious situation.	
INFO	Log.i()	Other than above, log information which is output to notify any remarkable changes or results in application state.	
DEBUG	Log.d()	Program's internal state information which needs to be output temporarily for analyzing the cause of specific bug when developing application.	Log information as per left is only for application developers. Therefore, this type of information should not be output in case of release version application.
VERBOSE	Log.v()	Log information which is not applied to any of above. Log information which application developer outputs for many purposes, is applied this. For example, in case of outputting server communication data to dump.	

For more details about logging method, refer to the following document.

Code Style Guidelines for Contributors / Log Sparingly

> https://source.android.com/setup/contribute/code-style#log-sparingly

4.8.3.3 DEBUG Log and VERBOSE Log Are Not Always Deleted Automatically

The following is quoted from the developer reference of android.util.Log class[28].

The order in terms of verbosity, from least to most is ERROR, WARN, INFO, DEBUG, VERBOSE. Verbose should never be compiled into an application except during development. Debug logs are compiled in but stripped at runtime. Error, warning and info logs are always kept.

After reading the above texts, some developers might have misunderstood the Log class behavior as per below.

- Log.v() call is not compiled when release build, VERBOSE log is never output.
- Log.v() call is compiled, but DEBUG log is never output when execution.

However, logging methods never behave in above ways, and all messages are output regardless of whether it is compiled with debug mode or release mode. If you read the document carefully, you will be able to realize that the gist of the document is not about the behavior of logging methods but basic policies for logging.

In this chapter, we introduced the sample code to get the expected result as described above by using ProGuard.

4.8.3.4 Remove Sensitive Information from Assembly

If you build the following code with ProGuard for the purpose of deleting Log.d() method, it is necessary to remember that ProGuard keeps the statement that construct the string for logging message (the first line of the code) even though it remove the statement of calling Log.d() method (the second line of the code).

```
String debug_info = String.format("%s:%s",
                                  "Sensitive information 1",
```

(continues on next page)

[28] https://developer.android.com/reference/android/util/Log.html

(continued from previous page)

```
                                    "Sensitive information 2");
if (BuildConfig.DEBUG) android.util.Log.d(TAG, debug_info);
```

The following disassembly shows the result of release build of the code above with ProGuard. Actually, there's no Log.d() call process, but you can see that character string consistence definition like "Sensitive information1" and calling process of String#format() method, are not deleted and still remaining there.

```
const-string v1, "%s:%s"
const/4 v2, 0x2
new-array v2, v2, [Ljava/lang/Object;
const/4 v3, 0x0
const-string v4, "Sensitive information 1"
aput-object v4, v2, v3
const/4 v3, 0x1
const-string v4, "Sensitive information 2"
aput-object v4, v2, v3
invoke-static {v1, v2}, Ljava/lang/String;->format(Ljava/lang/String;[Ljava/lang/
↪Object;)Ljava/lang/String;
move-result-object v0
```

Actually, it's not easy to find the particular part that disassembled APK file and assembled log output information as above. However, in some application which handles the very confidential information, this type of process should not be remained in APK file in some cases.

You should implement your application like below to avoid such a consequence of remaining the sensitive information in bytecode[29]. In release build, the following codes are deleted completely by the compiler optimization.

```
if (BuildConfig.DEBUG) {
    String debug_info = String.format("%s:%s",
                                      "Sensitive information 1",
                                      "Sensitive information 2");
    if (BuildConfig.DEBUG) android.util.Log.d(TAG, debug_info);
}
```

Besides, ProGuard cannot remove the log message of the following code("result:" + value).

```
Log.d(TAG, "result:" + value);
```

In this case, you can solve the problem in the following manner.

```
if (BuildConfig.DEBUG) Log.d(TAG, "result:" + value);
```

4.8.3.5 The Contents of Intent Is Output to LogCat

When using Activity, it's necessary to pay attention, since ActivityManager outputs the content of Intent to LogCat. Refer to "4.1.3.4. *Log Output When using Activities*".

4.8.3.6 Restrain Log which Is Output to System.out/err

System.out/err method outputs all messages to LogCat. Android could send some messages to System.out/err even if developers did not use these methods in their code, for example, in the following cases, Android sends stack trace to System.err method.

- When using Exception#printStackTrace()

[29] The previous sample code is enclosed in an if statement with BuildConfig.DEBUG as conditional expression. The if statement before the call to Log.d () is not necessary, but left as it is for comparison with the previous one.

- When it's output to System.err implicitly
 (When the exception is not caught by application, it's given to Exception#printStackTrace() by the system.)

You should handle errors and exceptions appropriately since the stack trace includes the unique information of the application.

We introduce a way of changing default output destination of System.out/err. The following code redirects the output of System.out/err method to nowhere when you build a release version application. However, you should consider whether this redirection does not cause a malfunction of application or system because the code temporarily overwrites the default behavior of System.out/err method. Furthermore, this redirection is effective only to your application and is worthless to system processes.

OutputRedirectApplication.java

```
/*
 * Copyright (C) 2012-2026 Japan Smartphone Security Association
 *
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 *
 *      http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */

package org.jssec.android.log.outputredirection;

import java.io.IOException;
import java.io.OutputStream;
import java.io.PrintStream;

import android.app.Application;

public class OutputRedirectApplication extends Application {

    // PrintStream which is not output anywhere
    private final PrintStream emptyStream = new PrintStream(new OutputStream() {
            public void write(int oneByte) throws IOException {
                // do nothing
            }
        });

    @Override
    public void onCreate() {
        // Redirect System.out/err to PrintStream which doesn't output anywhere,
        // when release build.

        // Save original stream of System.out/err
        PrintStream savedOut = System.out;
        PrintStream savedErr = System.err;

        // Once, redirect System.out/err to PrintStream which doesn't output
        // anywhere
        System.setOut(emptyStream);
        System.setErr(emptyStream);
```

(continues on next page)

(continued from previous page)

```
        // Restore the original stream only when debugging. (In release build,
        // the following 1 line is deleted byProGuard.)
        resetStreams(savedOut, savedErr);
    }

    // All of the following methods are deleted byProGuard when release.
    private void resetStreams(PrintStream savedOut, PrintStream savedErr) {
        System.setOut(savedOut);
        System.setErr(savedErr);
    }
}
```

```
AndroidManifest.xml
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
          >

  <application
      android:icon="@drawable/ic_launcher"
      android:label="@string/app_name"
      android:name=".OutputRedirectApplication"
      android:allowBackup="false" >
    <activity
        android:name=".LogActivity"
        android:label="@string/app_name"
        android:exported="true" >
      <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER" />
      </intent-filter>
    </activity>
  </application>
</manifest>
```

```
proguard-project.txt
# Prevent from changing class name and method name, etc.
-dontobfuscate

# In release build, delete call from Log.d()/v() automatically.
-assumenosideeffects class android.util.Log {
    public static int d(...);
    public static int v(...);
}

# In release build, delete resetStreams() automatically.
-assumenosideeffects class
    org.jssec.android.log.outputredirection.OutputRedirectApplication {
    private void resetStreams(...);
}
```

The difference of LogCat output between development version application (debug build) and release version application (release build) are shown as per below Fig. 4.8.3.

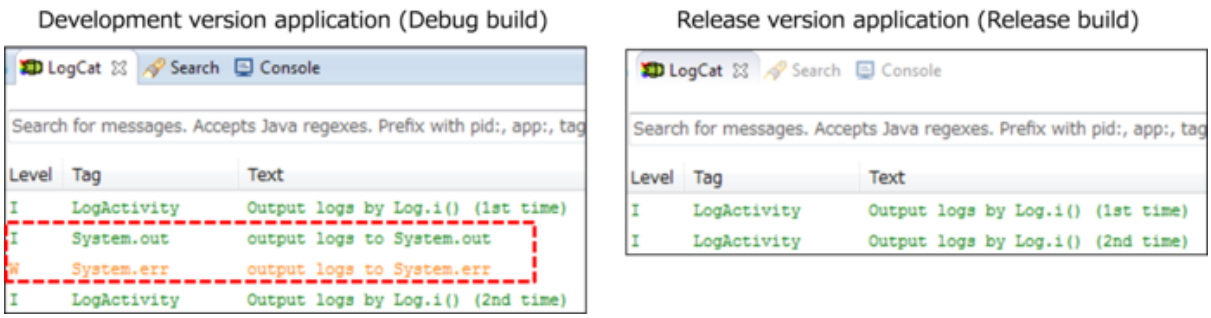


Fig. 4.8.3: Difference of System.out/err in LogCat output, between development application and release application.

4.9 Using WebView

WebView enables your application to integrate HTML/JavaScript content.

4.9.1 Sample Code

We need to take proper action, depending on what we'd like to show through WebView although we can easily show web site and html file by it. And also we need to consider risk from WebView's remarkable function; such as JavaScript-Java object bind.

Especially what we need to pay attention is JavaScript. (Please note that JavaScript is disabled as default. And we can enable it by WebSettings#setJavaScriptEnabled()). With enabling JavaScript, there is potential risk that malicious third party can get device information and operate your device.

The following is principle for application with WebView[30]:

(1) You can enable JavaScript if the application uses contents which are managed in house.

(2) You should NOT enable JavaScript other than the above case.

Fig. 4.9.1 shows flow chart to choose sample code according to content characteristic.

[30] Strictly speaking, you can enable JavaScript if we can say the content is safe. If the contents are managed in house, the contents should be guaranteed of security. And the company can secure them. In other words, we need to have business representation's decision to enable JavaScript for other company's contents. The contents which are developed by trusted partner might have security guarantee. But there is still potential risk. Therefore the decision is needed by responsible person.

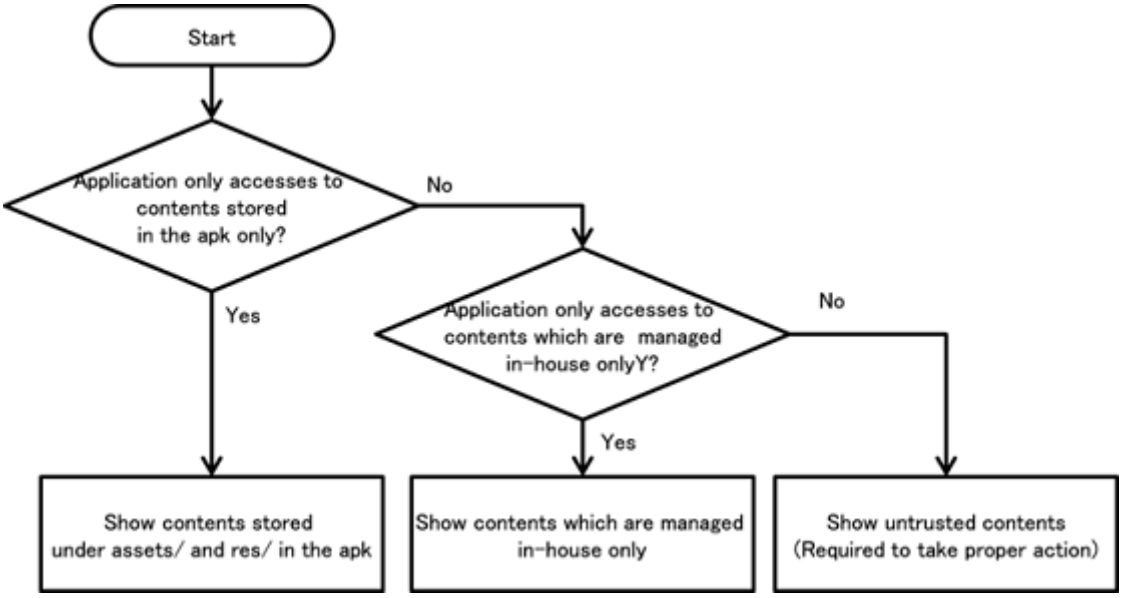


Fig. 4.9.1: Flow Figure to select Sample code of WebView

4.9.1.1 Show Only Contents Stored under assets/res Directory in the APK

You can enable JavaScript if your application shows only contents stored under assets/ and res/ directory in apk.

The following sample code shows how to use WebView to show contents stored under assets/ and res/.

Points:

1. Disable to access files (except files under assets/ and res/ in apk).
2. You may enable JavaScript.

WebViewAssetsActivity.java

```
/*
 * Copyright (C) 2012-2026 Japan Smartphone Security Association
 *
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 *
 *      http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */

package org.jssec.webview.assets;

import android.app.Activity;
import android.os.Bundle;
import android.webkit.WebSettings;
import android.webkit.WebView;

public class WebViewAssetsActivity extends Activity {
```

(continues on next page)

(continued from previous page)

```
    /**
     * Show contents in assets
     */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        WebView webView = (WebView) findViewById(R.id.webView);
        WebSettings webSettings = webView.getSettings();

        // *** POINT 1 *** Disable to access files (except files under assets/
        // and res/ in this apk)
        webSettings.setAllowFileAccess(false);

        // *** POINT 2 *** Enable JavaScript (Optional)
        webSettings.setJavaScriptEnabled(true);

        // Show contents which were stored under assets/ in this apk
        webView.loadUrl("file:///android_asset/sample/index.html");
    }
}
```

4.9.1.2 Show Only Contents which Are Managed In-house

You can enable JavaScript to show only contents which are managed in-house only if your web service and your Android application can take proper actions to secure both of them.

- Web service side actions:

As Fig. 4.9.2 shows, your web service can only refer to contents which are managed in-house. In addition, the web service is needed to take appropriate security action. Because there is potential risk if contents which your web service refers to may have risk; such as malicious attack code injection, data manipulation, etc.

Please refer to "4.9.2.1. *Enable JavaScript Only If Contents Are Managed In-house (Required)*".

- Android application side actions:

Using HTTPS, the application should establish network connection to your managed web service only if the certification is trusted.

The following sample code is an activity to show contents which are managed in-house.

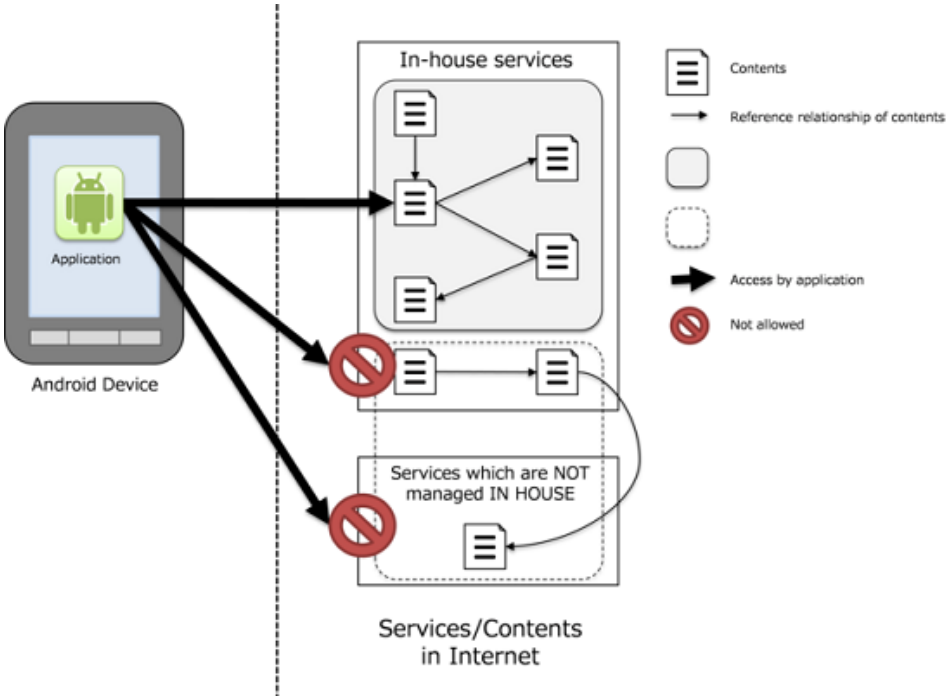


Fig. 4.9.2: Accessible contents and Non-accessible contents from application.

Points:

1. Handle SSL error from WebView appropriately.
2. (Optional) Enable JavaScript of WebView.
3. Restrict URLs to HTTPS protocol only.
4. Restrict URLs to in-house.

WebViewTrustedContentsActivity.java

```
/*
 * Copyright (C) 2012-2026 Japan Smartphone Security Association
 *
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 *
 *      http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */

package org.jssec.webview.trustedcontents;

import android.app.Activity;
import android.app.AlertDialog;
import android.content.DialogInterface;
import android.net.http.SslCertificate;
import android.net.http.SslError;
```

(continues on next page)

(continued from previous page)

```
import android.os.Bundle;
import android.webkit.SslErrorHandler;
import android.webkit.WebView;
import android.webkit.WebViewClient;

import java.text.SimpleDateFormat;

public class WebViewTrustedContentsActivity extends Activity {
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        WebView webView = (WebView) findViewById(R.id.webView);

        webView.setWebViewClient(new WebViewClient() {
            @Override
            public void onReceivedSslError(WebView view,
                                           SslErrorHandler handler,
                                           SslError error) {
                // *** POINT 1 *** Handle SSL error from WebView appropriately
                // Show SSL error dialog.
                AlertDialog dialog = createSslErrorDialog(error);
                dialog.show();

                // *** POINT 1 *** Handle SSL error from WebView appropriately
                // Abort connection in case of SSL error
                // Since, there may be some defects in a certificate like
                // expiration of validity, or it may be man-in-the-middle attack.
                handler.cancel();
            }
        });

        // *** POINT 2 *** Enable JavaScript (optional)
        // in case to show contents which are managed in house.
        webView.getSettings().setJavaScriptEnabled(true);

        // *** POINT 3 *** Restrict URLs to HTTPS protocol only
        // *** POINT 4 *** Restrict URLs to in-house
        webView.loadUrl("https://url.to.your.contents/");
    }

    private AlertDialog createSslErrorDialog(SslError error) {
        // Error message to show in this dialog
        String errorMsg = createErrorMessage(error);
        // Handler for OK button
        DialogInterface.OnClickListener onClickOk =
            new DialogInterface.OnClickListener() {
            @Override
            public void onClick(DialogInterface dialog, int which) {
                setResult(RESULT_OK);
            }
        };
        // Create a dialog
        AlertDialog dialog = new AlertDialog.Builder
            (WebViewTrustedContentsActivity.this).setTitle("SSL connection error")
            .setMessage(errorMsg).setPositiveButton("OK", onClickOk)
```

(continues on next page)

(continued from previous page)

```
            .create();
        return dialog;
    }

    private String createErrorMessage(SslError error) {
        SslCertificate cert = error.getCertificate();
        SimpleDateFormat dateFormat = new SimpleDateFormat("yyyy/MM/dd HH:mm:ss");
        StringBuilder result = new StringBuilder().append("The site's␣
↪certification is NOT valid. Connection was disconnected.\n\nError:\n");
        switch (error.getPrimaryError()) {
        case SslError.SSL_EXPIRED:
            result.append("The certificate is no longer valid.\n\nThe expiration␣
↪date is ").append(dateFormat.format(cert.getValidNotAfterDate()));
            return result.toString();
        case SslError.SSL_IDMISMATCH:
            result.append("Host name doesn't match. \n\nCN=").append(cert.
↪getIssuedTo().getCName());
            return result.toString();
        case SslError.SSL_NOTYETVALID:
            result.append("The certificate isn't valid yet.\n\nIt will be valid␣
↪from ").append(dateFormat.format(cert.getValidNotBeforeDate()));
            return result.toString();
        case SslError.SSL_UNTRUSTED:
            result.append("Certificate Authority which issued the certificate is␣
↪not reliable.\n\nCertificate Authority\n").append(cert.getIssuedBy().getDName());
            return result.toString();
        default:
            result.append("Unknown error occured. ");
            return result.toString();

        }
    }
}
```

4.9.1.3 Show Contents which Are Not Managed In-house

Don't enable JavaScript if your application shows contents which are not managed in house because there is potential risk to access to malicious content.

The following sample code is an activity to show contents which are not managed in-house.

This sample code shows contents specified by URL which user inputs through address bar. Please note that JavaScript is disabled and connection is aborted when SSL error occurs. The error handling is the same as "4.9.1.2. *Show Only Contents which Are Managed In-house*" for the details of HTTPS communication. Please refer to "5.4. *Communicating via HTTPS*" for the details also.

Points:

1. Handle SSL error from WebView appropriately.
2. Disable JavaScript of WebView.

WebViewUntrustActivity.java

```
/*
 * Copyright (C) 2012-2026 Japan Smartphone Security Association
 *
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
```

(continues on next page)

(continued from previous page)

```
 * You may obtain a copy of the License at
 *
 *      http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */

package org.jssec.webview.untrust;

import android.app.Activity;
import android.app.AlertDialog;
import android.content.DialogInterface;
import android.graphics.Bitmap;
import android.net.http.SslCertificate;
import android.net.http.SslError;
import android.os.Bundle;
import android.view.View;
import android.webkit.SslErrorHandler;
import android.webkit.WebView;
import android.webkit.WebViewClient;
import android.widget.Button;
import android.widget.EditText;

import java.text.SimpleDateFormat;

public class WebViewUntrustActivity extends Activity {
    /*
     * Show contents which are NOT managed in-house (Sample program works as a
     * simple browser)
     */

    private EditText textUrl;
    private Button buttonGo;
    private WebView webView;

    // Activity definition to handle any URL request
    private class WebViewUnlimitedClient extends WebViewClient {

        @Override
        public boolean shouldOverrideUrlLoading(WebView webView, String url) {
            webView.loadUrl(url);
            textUrl.setText(url);
            return true;
        }

        // Start reading Web page
        @Override
        public void onPageStarted(WebView webview, String url, Bitmap favicon) {
            buttonGo.setEnabled(false);
            textUrl.setText(url);
        }
```

(continues on next page)

(continued from previous page)

```
        // Show SSL error dialog
        // And abort connection.
        @Override
        public void onReceivedSslError(WebView webview,
                                       SslErrorHandler handler, SslError error) {

            // *** POINT 1 *** Handle SSL error from WebView appropriately
            AlertDialog errorDialog = createSslErrorDialog(error);
            errorDialog.show();
            handler.cancel();
            textUrl.setText(webview.getUrl());
            buttonGo.setEnabled(true);
        }

        // After loading Web page, show the URL in EditText.
        @Override
        public void onPageFinished(WebView webview, String url) {
            textUrl.setText(url);
            buttonGo.setEnabled(true);
        }
    }

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);

        webView = (WebView) findViewById(R.id.webview);
        webView.setWebViewClient(new WebViewUnlimitedClient());

        // *** POINT 2 *** Disable JavaScript of WebView
        // Explicitly disable JavaScript even though it is disabled by default.
        webView.getSettings().setJavaScriptEnabled(false);

        webView.loadUrl(getString(R.string.texturl));
        textUrl = (EditText) findViewById(R.id.texturl);
        buttonGo = (Button) findViewById(R.id.go);
    }

    public void onClickButtonGo(View v) {
        webView.loadUrl(textUrl.getText().toString());
    }

    private AlertDialog createSslErrorDialog(SslError error) {
        // Error message to show in this dialog
        String errorMsg = createErrorMessage(error);
        // Handler for OK button
        DialogInterface.OnClickListener onClickOk =
            new DialogInterface.OnClickListener() {
            @Override
            public void onClick(DialogInterface dialog, int which) {
                setResult(RESULT_OK);
            }
        };
        // Create a dialog
        AlertDialog dialog =
```

(continues on next page)

(continued from previous page)

```
            new AlertDialog.Builder(WebViewUntrustActivity.this)
                .setTitle("SSL connection error")
                .setMessage(errorMsg).setPositiveButton("OK", onClickOk)
                .create();
        return dialog;
    }

    private String createErrorMessage(SslError error) {
        SslCertificate cert = error.getCertificate();
        SimpleDateFormat dateFormat = new SimpleDateFormat("yyyy/MM/dd HH:mm:ss");
        StringBuilder result = new StringBuilder().append("The site's␣
↪certification is NOT valid. Connection was disconnected.\n\nError:\n");
        switch (error.getPrimaryError()) {
        case SslError.SSL_EXPIRED:
            result.append("The certificate is no longer valid.\n\nThe expiration␣
↪date is ").append(dateFormat.format(cert.getValidNotAfterDate()));
            return result.toString();
        case SslError.SSL_IDMISMATCH:
            result.append("Host name doesn't match. \n\nCN=").append(cert.
↪getIssuedTo().getCName());
            return result.toString();
        case SslError.SSL_NOTYETVALID:
            result.append("The certificate isn't valid yet.\n\nIt will be valid␣
↪from ").append(dateFormat.format(cert.getValidNotBeforeDate()));
            return result.toString();
        case SslError.SSL_UNTRUSTED:
            result.append("Certificate Authority which issued the certificate is␣
↪not reliable.\n\nCertificate Authority\n").append(cert.getIssuedBy().getDName());
            return result.toString();
        default:
            result.append("Unknown error occured. ");
            return result.toString();
        }
    }
}
```

4.9.2 Rule Book

Comply with following rule when you need to use WebView.

1. *Enable JavaScript Only If Contents Are Managed In-house (Required)*
2. *Use HTTPS to Communicate to Servers which Are Managed In-house (Required)*
3. *Verify That the URL Received from Others Such as Through Intent Is the Expected URL (Required)*
4. *Handle SSL Error Properly (Required)*

4.9.2.1 Enable JavaScript Only If Contents Are Managed In-house (Required)

What we have to pay attention on WebView is whether we enable the JavaScript or not. As principle, we can only enable the JavaScript only IF the application will access to services which are managed in-house. And you must not enable the JavaScript if there is possibility to access services which are not managed in-house.

Services managed In-house

In case that application accesses contents which are developed IN HOUSE and are distributed through servers which are managed IN HOUSE, we can say that the contents are ONLY modified by your company. In addition, it is also needed that each content refers to only contents stored in the servers which have proper security.

In this scenario, we can enable JavaScript on the WebView. Please refer to "4.9.1.2. *Show Only Contents which Are Managed In-house*" also.

And you can also enable JavaScript if your application shows only contents stored under assets/ and res/ directory in the apk. Please refer to "4.9.1.1. *Show Only Contents Stored under assets/res Directory in the APK*" also.

Services unmanaged in-house

You must NOT think you can secure safety on contents which are NOT managed IN HOUSE. Therefore you have to disable JavaScript. Please refer to "4.9.1.3. *Show Contents which Are Not Managed In-house*".

In addition, you have to disable JavaScript if the contents are stored in external storage media; such as microSD because other application can modify the contents.

4.9.2.2 Use HTTPS to Communicate to Servers which Are Managed In-house (Required)

You have to use HTTPS to communicate to servers which are managed in-house because there is potential risk of spoofing the services by malicious third party.

Please refer to both "4.9.2.4. *Handle SSL Error Properly (Required)*", and "5.4. *Communicating via HTTPS*".

4.9.2.3 Verify That the URL Received from Others Such as Through Intent Is the Expected URL (Required)

Implementation to receive an Intent from other application and display the URL provided in the Intent parameter on the WebView is common on many applications. If the URL provided at this point is to be displayed without verifying the expected URL, malicious websites such as phishing websites may be displayed on the WebView. The problem with this implementation is that unspecified URLs that are not guaranteed to be safe may become displayed. This may result in damages even if the JavaScript of the WebView is disabled.

To verify that the URL provided as the parameter of the Intent is the expected URL, there is a method to store the URL whitelist to display in the application in advance. Safety can be secured by displaying only the URL that matches this whitelist on the WebView. In addition, the URL to be registered to the whitelist must be HTTPS.

```
WebViewActivity.java
    // Get the white list from the resource file
    String[] allowList = getResources().getStringArray(R.array.allow_url_list);

    // Check the URL domain received from Intent is included the white list
    Uri uri = Uri.parse("the URL received from Intent");
    for (String str : allowList) {
        if (uri.getScheme().equals("https") && uri.getHost().equals(str)) {
            webView.loadUrl(uri.toString());
    }
```

To show the received URL on a WebView with JavaScript enabled, you must additionally verify that this URL is managed in-house.

Sample code in the section "4.9.1.2. *Show Only Contents which Are Managed In-house*" uses the fixed value URL to show contents which are managed in-house, to secure safety.

4.9.2.4 Handle SSL Error Properly (Required)

You have to terminate the network communication and inform error notice to user when SSL error happens on HTTPS communication.

SSL error shows invalid server certification risk or MTIM (man-in-the-middle attack) risk. Please note that WebView has NO error notice mechanism regarding SSL error. Therefore your application has to show the error notice to inform the risk to the user. Please refer to sample code in the section of "4.9.1.2. *Show Only Contents which Are Managed In-house*", and "4.9.1.3. *Show Contents which Are Not Managed In-house*".

In addition, your application MUST terminate the communication with the error notice.

In other words, you MUST NOT do following.

- Ignore the error to keep the transaction with the service.
- Retry HTTP communication instead of HTTPS.

Please refer to the detail described in "5.4. *Communicating via HTTPS*".

WebView's default behavior is to terminate the communication in case of SSL error. Therefore what we need to add is to show SSL error notice. And then we can handle SSL error properly.

4.9.3 Advanced Topics

4.9.3.1 Vulnerability caused by addJavascriptInterface() at Android versions 4.1 or earlier

Android versions under 4.2API Level 17 have a vulnerability caused by addJavascriptInterface(), which could allow attackers to call native Android methods (Java) via JavaScript on WebView.

As explained in "4.9.2.1. *Enable JavaScript Only If Contents Are Managed In-house (Required)*", JavaScript must not be enabled if the services could access services out of in-house control.

In Android 4.2API Level 17 or later, the measure of the vulnerability has been taken to limit access from JavaScript to only methods with @JavascriptInterface annotation on Java source codes instead of all methods of Java objects injected. However it is necessary to disable JavaScript if the services could access services out of in-house control as mentioned in "4.9.2.1.".

4.9.3.2 Issue caused by file scheme

In case of using WebView with default settings, all files that the app has access rights can be accessed to by using the file scheme in web pages regardless of the page origins. For example, a malicious web page could access the files stored in the app's private directory by sending a request to the uri of a private file of the app with the file scheme.

A countermeasure is to disable JavaScript as explained in "4.9.2.1. *Enable JavaScript Only If Contents Are Managed In-house (Required)*" if the services could access services out of in-house control. Doing that is to protect against sending the malicious file scheme request.

Also in case of Android 4.1 (API Level 16) or later, setAllowFileAccessFromFileURLs() and setAllowUniversalAccessFromFileURLs() can be used to limit access via the file scheme.

Disabling the file scheme

webView = (WebView) findViewById(R.id.webview); webView.setWebViewClient(new WebViewUnlimitedClient()); WebSettings settings = webView.getSettings(); settings.setAllowUniversalAccessFromFileURLs(false); settings.setAllowFileAccessFromFileURLs(false);

4.9.3.3 Specifying a Sender Origin When Using Web Messaging

Android 6.0 (API Level 23) adds an API for realizing HTML5 Web Messaging. Web Messaging is a framework defined in HTML5 for sending and receiving data between different browsing contexts[31].

The postWebMessage() method added to the WebView class is a method for processing data transmissions via the Cross-domain messaging protocol defined by Web Messaging.

[31] https://www.w3.org/TR/webmessaging/

This method sends a message object—specified by its first parameter—from the browsing context that has been read into WebView; however, in this case it is necessary to specify the origin of the sender as the second parameter. If the specified origin[32] does not agree with the origin in the sender context, the message will not be sent. By placing restrictions on the sender origin in this way, this mechanism aims to prevent the passing of messages to unintended senders.

However, it is important to note that wildcards may be specified as the origin in the postWebMessage() method[33]. If wildcards are specified, the sender origin of the message is not checked, and the message may be sent from any arbitrary origin. In a situation in which malicious content has been read into WebView, various types of harm or damage may result if important messages are sent without origin restrictions. Thus, when using WebView for Web messaging, it is best to specify explicitly a specific origin in the postWebMessage() method.

4.9.3.4 Safe Browsing in WebView

Safe Browsing is a service provided by Google that displays a warning page when the user tries to access a malware page, phishing site, or other unsafe web page.

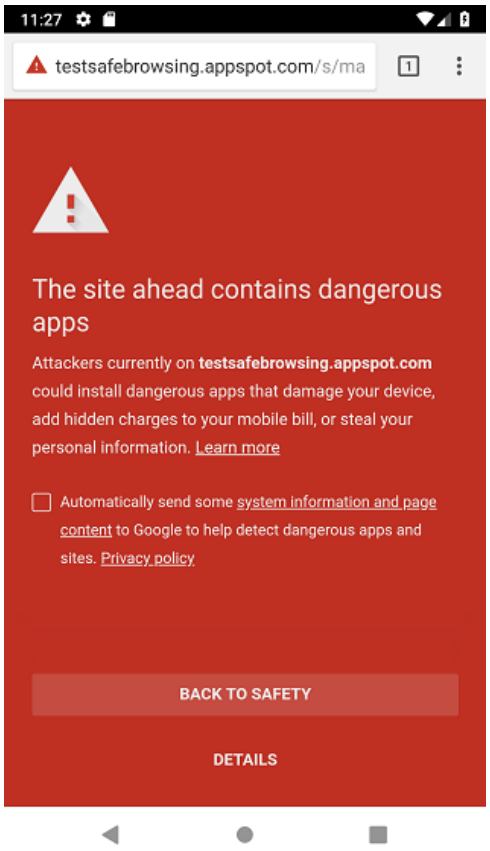


Fig. 4.9.3: Warning page displayed when attempting to access an unsafe web page in Chrome for Android

Currently, the Safe Browsing function can be used not only in Chrome for Android and other browser applications, but also in the WebView used in applications. However, careful attention is needed because the components that can be used for WebView vary depending on the Android OS version of the system, as a result, the degree of support for Safe Browsing also varies. Support for standard WebView and Safe Browsing by Android OS versions are shown in the following table.

[32] An “origin” is a URL scheme together with a host name and port number. For the detailed definition see http://tools.ietf.org/html/rfc6454.
[33] Note that Uri.EMPTY and Uri.parse(“”) function as wildcards (at the time of writing the September 1, 2016 version).

Table 4.9.1: Android OS version and standard WebView support

Android OS version	Android standard WebView	Relation with OS	Adapting to Safe Browsing
Android 7.0 or later	Chrome for Android (Chromium base)	**Independent**	**OK**
Android 5.0 - 6.0	Android System WebView (Chromium base)	**Independent**	**OK**
Android 4.4	OS embedded WebView (Chromium base)	Embedded	No
Android 4.3 or earlier	OS embedded WebView	Embedded	No

Before Android 4.3 (API level 18), a WebView that did not include the Safe Browsing function was incorporated into the OS, and this was changed in Android 4.4 (API level 19) so that WebView included the Safe Browsing function. Even so, care is needed because the version is old, and it does not support use of the Safe Browsing function in the WebView of applications.

The capability to use the Safe Browsing function in applications started from Android 5.0 (API level 21) when WebView was separated from the OS and became updated as an application.

Starting from WebView 66, Safe Browsing is enabled by default, and no special settings are required at the application side. However, it is possible that Safe Browsing may not be enabled by default for some WebView versions if the user did not update WebView or if the standard WebView in the "Set WebView implementation" option for developers was changed from the default. And so, if Safe Browsing is used, it must be explicitly enabled as shown below.

Settings for enabling Safe Browsing in AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest package="...">
    <application>
        ...

        <meta-data
            android:name="android.webkit.WebView.EnableSafeBrowsing"
            android:value="true" />
    </application>
</manifest>
```

Also, in Android 8.0 (API level 26), several APIs for Safe Browsing were added.

The setSafeBrowsingEnabled(boolean enabled) added in the WebSettings class is a setting method for dynamically enabling or disabling each WebView instance. Before Android 8.0 (API level 26), the Safe Browsing function was enabled or disabled by settings in AndroidManifest, but this could only make settings for all WebViews in an application. The setSafeBrowsingEnabled(boolean enabled) can be used to allow dynamic enable/disable switching for each WebView instance.

```
if (url == IN_HOUSE_MANAGEMENT_CONTENT_URL) {
    // (ex.) because in-house contents are detectable by Safe Browsing,
    // diable it temporarily
    webView.getSettings().setSafeBrowsingEnabled(false);
} else {
    // normally, it should be enabled
    webView.getSettings().setSafeBrowsingEnabled(true);
}
```

Also, in Android 8.1 (API level 27), classes and APIs for Safe Browsing were added. These enable specifying of the Safe Browsing initialization process, settings for responses taken when accessing an unsafe web page, setting of a whitelist for excluding specific sites from Safe Browsing, and more.

The startSafeBrowsing() added in the WebView class is a method that calls the Safe Browsing initialization process for WebView components used for the WebView in applications. The initialization result is passed to the callback

object that is passed by the 2nd argument, and so if initialization fails, and false is passed to the callback object, responses such as disabling WebView or not loading the URL are recommended.

```
// because the Safe Browsing is not supported before Android 8.1,
// real implementations need to check Android OS version of the device
WebView.startSafeBrowsing(this, new ValueCallback<Boolean>() {
    @Override
    public void onReceiveValue(Boolean result) {
        mSafeBrowsingIsInitialized = true;
        if (result) {
            Log.i("WebView SafeBrowsing", "Initialized SafeBrowsing!");
        } else {
            Log.w("WebView SafeBrowsing", "SafeBrowsing initialization failed...");
            // when the initialization failed, Safe Browsing might not work
            // properly in this case, it is advisable to disable WebView
        }
    }
});
```

Similarly, the setSafeBrowsingWhitelist() added in the WebView class is a method that sets host names and IP addresses that are excluded from Safe Browsing in a whitelist format. When a list of the host names and IP addresses to be excluded from Safe Browsing is passed as an argument, no verification is conducted using Safe Browsing when they are accessed.

```
// setting the white list of the pair of host name and Ip address which is
// excluded from Safe Browsing
// (ex.) because in-house contents are detectable by Safe Browsing, register
// them to white list
WebView.setSafeBrowsingWhitelist(new ArrayList<>(Arrays.asList( IN_HOUSE_
↪MANAGEMENT_CONTENT_HOSTNAME )),
    new ValueCallback<Boolean>() {
        @Override
        public void onReceiveValue(Boolean aBoolean) {
            Log.i("WebView SafeBrowsing", "Whitelisted " + aBoolean.toString());
        }
    });
```

The onSafeBrowsingHit() added in the WebClient class is a callback function that is called back when it is determined that a URL accessed in a WebView where Safe Browsing is enabled is an unsafe web page. The object of the WebView that accessed the unsafe web page is passed to the 1st argument, WebResourceRequest is passed to the 2nd argument, the type of threat is passed to the 3rd argument, and the SafeBrowsingResponse object for setting the response when determining that a page is unsafe is passed to the 4th argument.

The response when using the SafeBrowsingResponse object can be selected from the three options below.

- backToSafety(boolean report): Returns to the previous page without displaying a warning (If no previous page is available, a blank page is displayed.)
- proceed(boolean report): Ignores the warning and displays the web page.
- showInterstitial(boolean allowReporting): Displays the warning page (default response)

For backToSafety() and proceed(), an argument can be used to set whether a report is sent to Google, and an argument can be set for showInterstitial() to display "a checkbox for selecting whether a report is sent to Google".

```
public class MyWebViewClient extends WebViewClient {

    // When Safe Browsing function is enabled, accessing unsafe web page will
    // cause this callback to be ivoked
    @Override
```

(continues on next page)

(continued from previous page)

```
    public void onSafeBrowsingHit(WebView view, WebResourceRequest request,
                                  int threatType, SafeBrowsingResponse callback) {
        // Display warning page with a check box which selects
        // "Send report to Google" or not (Recommended)
        callback.showInterstitial(true);
        // Without displaying warning page, return back to the safe page,
        // and send a report to Google (Recommended)
        callback.backToSafety(true);
        // Ignoring the warning, access to the page, and send a report to
        // Google (Not recommended)
        callback.proceed(false);
    }
}
```

No Android Support Library is available that supports these classes and APIs. For this reason, to operate applications using these classes and APIs in systems that are below API level 26 or 27, the processes must be separated based on the version or similar measures are required.

Sample code is shown below for handling of access to unsafe web pages when Safe Browsing is used in WebView.

AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
          >

  <uses-permission android:name="android.permission.INTERNET" />

  <application
      android:allowBackup="false"
      android:icon="@mipmap/ic_launcher"
      android:label="@string/app_name"
      android:theme="@style/AppTheme"
      android:networkSecurityConfig="@xml/network_security_config">
    <activity
        android:name=".MainActivity"
        android:exported="true"
        android:label="@string/app_name">
      <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER" />
      </intent-filter>
    </activity>

    <!-- Explicitly enable the Safe Browsing function of WebView in the␣
↪application process -->
    <meta-data
        android:name="android.webkit.WebView.EnableSafeBrowsing"
        android:value="true" />
  </application>
</manifest>
```

MainActivity.java

```
/*
 * Copyright (C) 2012-2026 Japan Smartphone Security Association
 *
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
```

(continues on next page)

(continued from previous page)

```
 * You may obtain a copy of the License at
 *
 *      http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */
package org.jssec.android.webview.safebrowsing;

import androidx.appcompat.app.AppCompatActivity;

import android.os.Bundle;
import android.util.Log;
import android.view.View;
import android.webkit.ValueCallback;
import android.webkit.WebView;

import java.util.ArrayList;
import java.util.Arrays;

public class MainActivity extends AppCompatActivity {

    private boolean mSafeBrowsingIsInitialized;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        findViewById(R.id.button1).setOnClickListener(setWhiteList);
        findViewById(R.id.button2).setOnClickListener(reload);

        final WebView webView = findViewById(R.id.webView);
        webView.setWebViewClient(new MyWebViewClient());

        mSafeBrowsingIsInitialized = false;
        // Because Safe Browsing is not supported on a device below Android 8.1,
        // real implementation needs to check Android OS version of the device
        WebView.startSafeBrowsing(this, new ValueCallback<Boolean>() {
            @Override
            public void onReceiveValue(Boolean result) {
                mSafeBrowsingIsInitialized = true;
                if (result) {
                    Log.i("WebView SafeBrowsing", "Initialized SafeBrowsing!");
                    webView.loadUrl("http://testsafebrowsing.appspot.com/s/malware.
↪html");
                } else {
                    Log.w("WebView SafeBrowsing", "SafeBrowsing initialization␣
↪failed...");
                    // When the initilization failed, Safe Browsing might not work
                    // properly. In this case, it is advaisable not to load URL.
                }
            }
```

(continues on next page)

(continued from previous page)

```
        });
    }

    View.OnClickListener setWhiteList = new View.OnClickListener() {
        @Override
        public void onClick(View view) {
            // Set the white list of the pair of host name and Ip address which
            // is excluded from Safe Browsing
            WebView.setSafeBrowsingWhitelist(new ArrayList<>(Arrays.asList(
↪"testsafebrowsing.appspot.com")), new ValueCallback<Boolean>() {
                @Override
                public void onReceiveValue(Boolean aBoolean) {
                    Log.i("WebView SafeBrowsing", "Whitelisted " + aBoolean.
↪toString());
                }
            });
        }
    };

    View.OnClickListener reload = new View.OnClickListener() {
        @Override
        public void onClick(View view) {
            final WebView webView = findViewById(R.id.webView);
            webView.reload();
        }
    };
}
```

MyWebViewClient.java

```
/*
 * Copyright (C) 2012-2026 Japan Smartphone Security Association
 *
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 *
 *      http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */
package org.jssec.android.webview.safebrowsing;

import android.webkit.SafeBrowsingResponse;
import android.webkit.WebResourceRequest;
import android.webkit.WebView;
import android.webkit.WebViewClient;
import android.widget.Toast;

public class MyWebViewClient extends WebViewClient {

    // When Safe Browsing is enabled, accessing unsafe Web page will cause this
    // callback to be invoked
```

(continues on next page)

(continued from previous page)

```
    @Override
    public void onSafeBrowsingHit(WebView view, WebResourceRequest request,
                                  int threatType, SafeBrowsingResponse callback) {
        // Do not display warningpage, and return back to safe page
        callback.backToSafety(true);
        Toast.makeText(view.getContext(), "Because the visiting web page is␣
↪suspicious to be a malware site, we are returning back to the safe page.", Toast.
↪LENGTH_LONG).show();
    }
}
```

4.10 Using Notifications

Android offers the Notification feature for sending messages to end users. Using a Notification causes a region known as a status bar to appear on the screen, inside which you may display icons and messages.

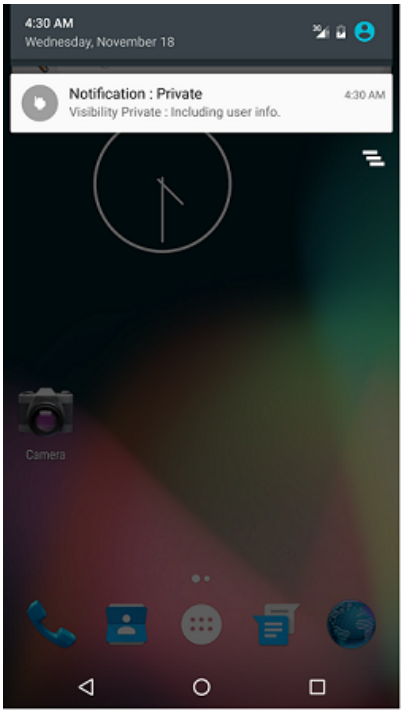


Fig. 4.10.1: An example of a Notification

The communication functionality of Notifications is enhanced in Android 5.0 (API Level 21) to allow messages to be displayed via Notifications even when the screen is locked, depending on user and application settings. However, incorrect use of Notifications runs the risk that private information—which should only be shown to the terminal user herself—may be seen by third parties. For this reason, this functionality must be implemented with careful attention paid to privacy and security.

The possible values for the Visibility option and the corresponding behavior of Notifications is summarized in the following table.

Table 4.10.1: Possible visibility values and behavior of Notifications

Visibility value	Behavior of Notivications
Public	Notifications are displayed on all locked screens.
Private	Notifications are displayed on all locked screens; however, on locked screens that have been password-protected (secure locks), fields such as the title and text of the Notification are hidden (replaced by publicly-releasable messages in which private information is hidden).
Secret	Notifications are not displayed on locked screens that are protected by passwords or other security measures (secure locks). (Notifications are displayed on locked screens that do not involve secure locks.)

4.10.1 Sample Code

When a Notification contains private information regarding the terminal user, a message from which the private information has been excluded must be prepared and added to be displayed in the event of a locked screen.

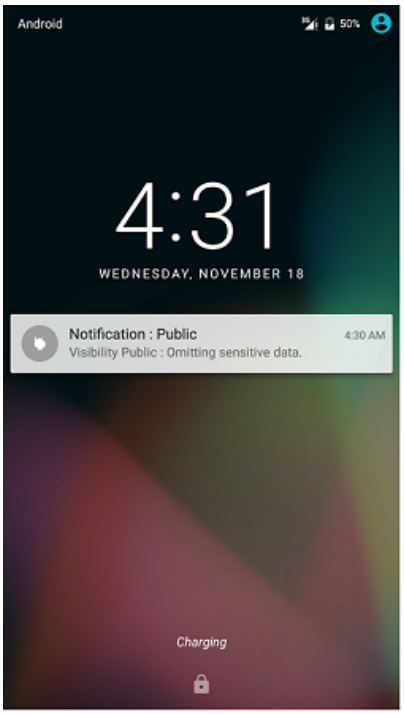


Fig. 4.10.2: A notification on a locked screen

Sample code illustrating the proper use of Notifications for messages containing private data is shown below.

Points:

1. When using Notifications for messages containing private data, prepare a version of the Notification that is suitable for public display (to be displayed when the screen is locked).
2. Do not include private information in Notifications prepared for public display (displayed when the screen is locked).
3. Explicitly set Visibility to Private when creating Notifications.
4. When Visibility is set to Private, Notifications may contain private information.

```
VisibilityPrivateNotificationActivity.java
/*
 * Copyright (C) 2012-2026 Japan Smartphone Security Association
```

(continues on next page)

(continued from previous page)

```
 *
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 *
 *      http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */

package org.jssec.notification.visibilityPrivate;

import static android.app.PendingIntent.FLAG_MUTABLE;

import android.app.Activity;
import android.app.Notification;
import android.app.NotificationChannel;
import android.app.NotificationManager;
import android.app.PendingIntent;
import android.app.Person;
import android.app.RemoteInput;
import android.content.Context;
import android.content.Intent;
import android.graphics.drawable.Icon;
import android.net.Uri;
import android.os.Build;
import android.os.Bundle;
import android.view.View;

public class VisibilityPrivateNotificationActivity extends Activity {
    /**
     * Display a private Notification
     */
    private final int mNotificationId = 0;

    public static final String DEFAULT_CHANNEL = "default_channel";
    public static final String SENDER_NAME = "Sender Name";

    public static final String REMOTE_REPLY = "remote_reply";
    public static final String reply_choices[] = {"choice1", "choice2", "choice3"};
    public static final String REPLY_LABEL = "input reply";

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        if (Build.VERSION.SDK_INT >= 26) {
            // Channel is required for Notification from api level 26
            NotificationChannel default_channel =
                new NotificationChannel(DEFAULT_CHANNEL,
                    getString(R.string.notification_channel_default),
```

(continues on next page)

(continued from previous page)

```
                    NotificationManager.IMPORTANCE_DEFAULT);

            NotificationManager notificationManager = (NotificationManager) this.
↪getSystemService(Context.NOTIFICATION_SERVICE);
            notificationManager.createNotificationChannel(default_channel);
        }
    }

    public void onSendNotificationClick(View view) {

        // *** POINT 1 *** When preparing a Notification that includes private
        // information, prepare an additional Noficiation for public display
        // (displayed when the screen is locked).
        Notification.Builder publicNotificationBuilder;
        if (Build.VERSION.SDK_INT >= 26) {
            publicNotificationBuilder =
                new Notification.Builder(this, DEFAULT_CHANNEL)
                        .setContentTitle("Notification : Public");
        } else {
            publicNotificationBuilder =
                new Notification.Builder(this)
                        .setContentTitle("Notification : Public");
        }

        if (Build.VERSION.SDK_INT >= 21)
            publicNotificationBuilder
                .setVisibility(Notification.VISIBILITY_PUBLIC);
        // *** POINT 2 *** Do not include private information in Notifications
        // prepared for public display (displayed when the screen is locked).
        publicNotificationBuilder.setContentText("Visibility Public : Omitting␣
↪sensitive data.");
        publicNotificationBuilder.setSmallIcon(R.drawable.ic_launcher);
        Notification publicNotification = publicNotificationBuilder.build();

        // Construct a Notification that includes private information.
        Notification.Builder privateNotificationBuilder;
        if (Build.VERSION.SDK_INT >= 26) {
            privateNotificationBuilder =
                    new Notification.Builder(this, DEFAULT_CHANNEL)
                            .setContentTitle("Notification : Private");
        } else {
            privateNotificationBuilder =
                    new Notification.Builder(this)
                            .setContentTitle("Notification : Private");
        }

        // *** POINT 3 *** Explicitly set Visibility to Private when creating
        // Notifications.
        if (Build.VERSION.SDK_INT >= 21)
            privateNotificationBuilder
                .setVisibility(Notification.VISIBILITY_PRIVATE);
        // *** POINT 4 *** When Visibility is set to Private, Notifications may
        // contain private information.
        privateNotificationBuilder
            .setContentText("Visibility Private : Including user info.");
        privateNotificationBuilder.setSmallIcon(R.drawable.ic_launcher);
```

(continues on next page)

(continued from previous page)

```
        // When creating a Notification with Visibility=Private, we also create
        // and register a separate Notification with Visibility=Public for public
        // display.
        if (Build.VERSION.SDK_INT >= 21)
            privateNotificationBuilder
                .setPublicVersion(publicNotification);

        if (Build.VERSION.SDK_INT >= 28) {
            // Display resource sample_picture.png in Notification by
            // Notification.MessagingStyle.Message.setData()
            int resourceId = R.drawable.sample_picture_small;
            Uri imageUri = Uri.parse("android.resource://" +
                                     getApplicationContext().getPackageName() +
                                     "/" + resourceId);

            Person sender = new Person.Builder()
                    .setName(SENDER_NAME)
                    .setIcon(null)
                    .build();

            // Prepare Notification.MessagingStyle.Message and set the image with
            // setData()
            Notification.MessagingStyle.Message message =
                new Notification.MessagingStyle
                        .Message("Sample Picture", 0, sender)
                        .setData("image/png", imageUri);

            // Prepare Notification.MessagingStyle and set
            // Notification.MessagingStyle.Message that sets the image
            Notification.MessagingStyle message_style =
                    new Notification.MessagingStyle(sender)
                    .addMessage(message);

            // Set Notification.MessagingStyle to Notification
            privateNotificationBuilder.setStyle(message_style);
        }

        if (Build.VERSION.SDK_INT >= 28) {
            // Display reply options in Notification by
            // RemoteInput.Builder.setChoices()
            Intent intent =
                new Intent( getApplicationContext(), NotificationReceiver.class);

            PendingIntent pendingIntent =
                PendingIntent.getBroadcast(this, 0, intent, 0);

            RemoteInput remoteInput = new RemoteInput.Builder(REMOTE_REPLY)
                    .setLabel(REPLY_LABEL)
                    .setChoices(reply_choices)
                    .build();

            Icon icon = Icon.createWithResource(this, R.drawable.ic_launcher);

            Notification.Action actionReply =
                new Notification.Action.Builder(icon, REPLY_LABEL, pendingIntent)
                    .addRemoteInput(remoteInput)
```

(continues on next page)

(continued from previous page)

```
                    .setSemanticAction(Notification.Action.SEMANTIC_ACTION_REPLY)
                    .build();

            privateNotificationBuilder.addAction(actionReply);
        }

        Notification privateNotification = privateNotificationBuilder.build();
        //Although not implemented in this sample code, in many cases
        //Notifications will use setContentIntent(PendingIntent intent)
        //to ensure that an Intent is transmission when Notification
        //is clicked. In this case, it is necessary to take steps--depending
        //on the type of component being called--to ensure that the Intent
        //in question is called by safe methods (for example, by explicitly
        //using Intent). For information on safe methods for calling various
        //types of component, see the following sections.
        //4.1. Creating and using Activities
        //4.2. Sending and receiving Broadcasts
        //4.4. Creating and using Services

        NotificationManager notificationManager =
                (NotificationManager) this
                .getSystemService(Context.NOTIFICATION_SERVICE);
        notificationManager.notify(mNotificationId, privateNotification);
    }
}
```

4.10.2 Rule Book

When creating Notification, the following rules must be observed.

1. *Regardless of the Visibility setting, Notifications must not contain sensitive information (although private information is an exception) (Required)*
2. *Notifications with Visibility=Public must not contain private information (Required)*
3. *For Notifications that contain private information, Visibility must be explicitly set to Private or Secret (Required)*
4. *When using Notifications with Visibility=Private, create an additional Notification with Visibility=Public for public display (Recommended)*

4.10.2.1 Regardless of the Visibility setting, Notifications must not contain sensitive information (although private information is an exception) (Required)

On terminals using Android4.3 (API Level 18) or later, users can use the Settings window to grant apps permission to read Notifications. Apps granted this permission will be able to read all information in Notifications; for this reason, sensitive information must not be included in Notifications. (However, private information may be included in Notifications depending on the Visibility setting).

Information contained in Notifications may generally not be read by apps other than the app that sent the Notification. However, users may explicitly grant permission to certain user-selected apps to read all information in Notifications. Because only apps that have been granted user permission may read information in Notifications, there is nothing problematic about including private information on the user within the Notification. On the other hand, if sensitive information other than the user's private information (for example, secret information known only to the app developers) is include in a Notification, the user herself may attempt to read the information contained in the Notification and may grant applications permission to view this information as well; thus the inclusion of sensitive information other than private user information is problematic.

For specific methods and conditions, see Section "4.10.3.1. *On User-granted Permission to View Notifications*"

4.10.2.2 Notifications with Visibility=Public must not contain private information (Required)

When sending Notifications with Visibility=Public, private user information must not be included in the Notification. When a Notifications has the setting Visibility=Public, the information in the Notification is displayed even when the screen is locked. This is because such Notifications carry the risk that private information might be seen and stolen by a third party in physical proximity to the terminal.

```
VisibilityPrivateNotificationActivity.java
    // Prepare a Notification for public display (to be displayed on locked
    // screens) that does not contain sensitive information.
    Notification.Builder publicNotificationBuilder =
        new Notification.Builder(this).setContentTitle("Notification : Public");

    publicNotificationBuilder.setVisibility(Notification.VISIBILITY_PUBLIC);
    // Do not include private information in Notifications for public display
    // (to be displayed on locked screens).
    publicNotificationBuilder.setContentText("Visibility Public: sending␣
↪notification without sensitive information.");
    publicNotificationBuilder.setSmallIcon(R.drawable.ic_launcher);
```

Typical examples of private information include emails sent to the user, the user's location data, and other items listed in Section "5.5. *Handling privacy data*".

4.10.2.3 For Notifications that contain private information, Visibility must be explicitly set to Private or Secret (Required)

Terminals using Android 5.0 (API Level 21) or later will display Notifications even when the screen is locked. Thus, when the Notification contains private information, its Visibility flag should be set explicitly to "Private" or "Secret". This is to protect against the risk of private information contained in a Notification being displayed on a locked screen.

At present, the default value of Visibility is set to Private for Notifications, so the aforementioned risk will only arise if this flag is explicitly changed to Public. However, the default value of Visibility may change in the future; for this reason, and also for the purpose of clearly communicating one's intentions at all times when handling information, it is mandatory to set Visibility=Private explicitly for Notifications that contain private information.

```
VisibilityPrivateNotificationActivity.java
    // Create a Notification that includes private information.
    Notification.Builder priavteNotificationBuilder =
        new Notification.Builder(this).setContentTitle("Notification : Private");

    // *** POINT *** Explicitly set Visibility=Private when creating the
    // Notification.
    priavteNotificationBuilder.setVisibility(Notification.VISIBILITY_PRIVATE);
```

4.10.2.4 When using Notifications with Visibility=Private, create an additional Notification with Visibility=Public for public display (Recommended)

When communicating information via a Notification with Visibility=Private, it is desirable to create simultaneously an additional Notification, for public display, with Visibility=Public; this is to restrict the information displayed on locked screens.

If a public-display Notification is not registered together with a Visibility=Private notification, a default message prepared by the operating system will be displayed when the screen is locked. Thus there is no security problem in this case. However, for the purpose of clearly communicating one's intentions at all times when handling information, it is recommended that a public-display Notification be explicitly created and registered.

```
VisibilityPrivateNotificationActivity.java
    // Create a Notification that contains private information.
    Notification.Builder privateNotificationBuilder =
        new Notification.Builder(this).setContentTitle("Notification : Private");

    // *** POINT *** Explicitly set Visibility=Private when creating the
    // Notification.
    if (Build.VERSION.SDK_INT >= 21)
        privateNotificationBuilder.setVisibility(Notification.VISIBILITY_PRIVATE);
    // *** POINT *** Notifications with Visibility=Private may include private
    // information.
    privateNotificationBuilder
        .setContentText("Visibility Private : Including user info.");
    privateNotificationBuilder.setSmallIcon(R.drawable.ic_launcher);
    // When creating a Notification with Visibility=Private,
    // simultaneously create and register a public-display Notification with
    // Visibility=Public.
    if (Build.VERSION.SDK_INT >= 21)
        privateNotificationBuilder.setPublicVersion(publicNotification);
```

4.10.3 Advanced Topics

4.10.3.1 On User-granted Permission to View Notifications

As noted above in Section "4.10.2.1. *Regardless of the Visibility setting, Notifications must not contain sensitive information (although private information is an exception) (Required)*", on terminals using Android 4.3 (API Level 18) or later, certain user-selected apps that have been granted user permission may read information in all Notifications.

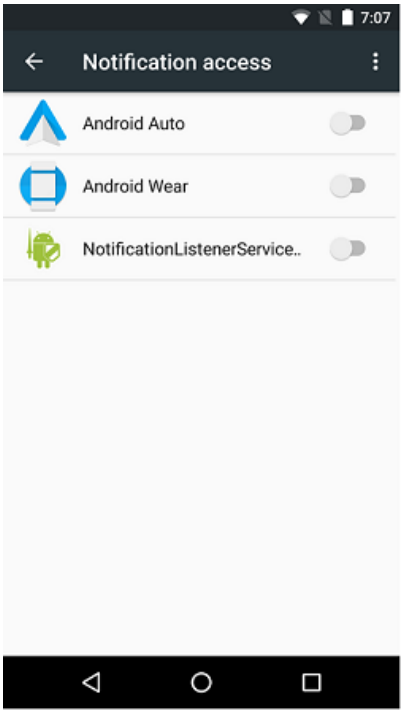


Fig. 4.10.3: "The Access to Notifications" window, from which Notification read controls may be configured

The following sample code illustrates the use of NotificationListenerService[34].

[34] In the testing environment at the time of the 12th edition, it was found that the sample does not work with the emulator Pixel 3 API 30

AndroidManifest.xml

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
          >

  <uses-permission android:name="android.permission.POST_NOTIFICATIONS"/>

  <application
      android:allowBackup="false"
      android:icon="@drawable/ic_launcher"
      android:label="@string/app_name" >
    <service android:name=".MyNotificationListenerService"
        android:exported="false"
        android:label="@string/app_name"
             android:permission="android.permission.BIND_NOTIFICATION_LISTENER_
↪SERVICE">
      <intent-filter>
        <action android:name=
                "android.service.notification.NotificationListenerService" />
      </intent-filter>
    </service>
  </application>
</manifest>
```

MyNotificationListenerService.java

```
/*
 * Copyright (C) 2012-2026 Japan Smartphone Security Association
 *
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 *
 *      http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */

package org.jssec.notification.notificationListenerService;

import android.app.Notification;
import android.service.notification.NotificationListenerService;
import android.service.notification.StatusBarNotification;
import android.util.Log;

public class MyNotificationListenerService extends NotificationListenerService {
    @Override
    public void onNotificationPosted(StatusBarNotification sbn) {
        // Notification is posted.
        outputNotificationData(sbn, "Notification Posted : ");
    }
```

(continues on next page)

attached to Android Studio 4.0.1. Since it has been confirmed that it works without problems on the Google Pixel 3 device, please be careful when executing the sample code.

(continued from previous page)

```
    @Override
    public void onNotificationRemoved(StatusBarNotification sbn) {
        // Notification is deleted.
        outputNotificationData(sbn, "Notification Deleted : ");
    }

    private void outputNotificationData(StatusBarNotification sbn, String prefix) {
        Notification notification = sbn.getNotification();
        int notificationID = sbn.getId();
        String packageName = sbn.getPackageName();
        long PostTime = sbn.getPostTime();

        String message = prefix + "Visibility :" + notification.visibility +
                         " ID : " + notificationID;
        message += " Package : " + packageName + " PostTime : " + PostTime;

        Log.d("NotificationListen", message);
    }
}
```

As discussed above, by using NotificationListenerService to obtain user permission it is possible to read Notifications. However, because the information contained in Notifications frequently includes private information on the terminal, care is required in handling such information.

4.10.3.2 Touch to Be Passed Through Specific Window

On Android 12 models and later, if the application is displaying an unsafe overlay, touches that pass through specific windows are blocked. This change of operation affects all applications running in Android 12 regardless of targetSdkVersion.

Examples of unsafe overlay displays are as follows.

- Overlay displays that require the SYSTEM_ALERT_WINDOW permission, such as windows that use the TYPE_APPLICATION_OVERLAY layer and the FLAG_NOT_TOUCHABLE flag
- Activity windows which use the FLAG_NOT_TOUCHABLE flag

However, in the following cases, touches that pass through are still available.

- If displaying overlays only within the application
- For trusted windows with TYPE_ACCESSIBILITY_OVERLAY, TYPE_INPUT_METHOD, etc. specified to the layers
- For invisible windows with route view set to GONE or INVISIBLE
- For windows with 0 set as the alpha value
- For windows with TYPE_APPLICATION_OVERLAY specified to the layer that has an alpha value lower than the specified value[35].

If an unsafe overlay is displayed on the application, it is recommended to use one of the following APIs based on the use case.

- Bubble

Bubble is a function that was added from Android 11 to make Notification use easier. The message notification appears as a Bubble on other applications when you have received messages, enabling you to display and reply to the message without having to switch to the application that received the message.

[35] Specified value indicates a value that can be acquired by InputManager.getMaximumObscuringOpacityForTouch(). If numerous windows are overlapping, the total alpha value must be lower than the specified value.

- Picture-in-Picture (PIP)

PIP is a function that displays content on a small window that is fixed to the corner of the screen even while you are moving around various applications or browsing through content on the main screen. Available on Android 8.0 and later.

- Notification

Notification is a standard method to provide reminders to users, messages from other people, and timely information from applications while reducing device usage to the minimum. Users can open the application by tapping the notification or run direct action from the notification.

- Snackbar

Snackbar is a notification function that displays messages for a short period of time while the application is running.

- Toast

A notification function that displays messages for a short period of time as on Snackbar. Use Toast if message display is required while the application is in the background. To grant permission to untrusted touches, run the following adb command on the terminal window.

```
# A specific app
adb shell am compat disable BLOCK_UNTRUSTED_TOUCHES com.example.app

# All apps
# If you'd still like to see a Logcat message warning when a touch would be
# blocked, use 1 instead of 0.
adb shell settings put global block_untrusted_touches 0
```

To return the operation to the default operation that blocks untrusted touches, run the following adb command on the terminal window.

```
# A specific app
adb shell am compat reset BLOCK_UNTRUSTED_TOUCHES com.example.app

# All apps
adb shell settings put global block_untrusted_touches 2
```

4.10.3.3 Runtime Permissions for Notifications

Starting from Android 13, the notification feature requires prior permission from the user. This change of operation affects all apps running on the Android 13 platform regardless of targetSdkVersion. The implementation method differs depending on the targetSdkVersion, and the implementation method for each case is described below.

For devices running targetSdkVersion 32 and below:

To use the notification feature, use createNotificationChannel to request permission from the user. A permission dialog appears when createNotificationChannel is executed, allowing the user to perform the following actions.

- Select “Allow”
- Select “Don’t allow”
- Close the dialog by swiping without pressing either button.

The following is an implementation example of an app running as a foreground service.

```
NotificationManager notificationManager = (NotificationManager)context.
↪getSystemService(Context.NOTIFICATION_SERVICE);

NotificationChannel channel = new NotificationChannel(defaultId, name ,␣
↪importance);
```

(continues on next page)

(continued from previous page)

```
if(notificationManager != null){
    notificationManager.createNotificationChannel(channel);

    Notification notification = new Notification.Builder(context, defaultId)
            .setContentTitle(name)
            .setSmallIcon(android.R.drawable.ic_media_play)
            .setContentText(name)
            .setAutoCancel(true)
            .setContentIntent(pendingIntent)
            .setWhen(System.currentTimeMillis())
            .build();

    startForeground(1, notification);
```

The permission dialog when the above code is executed on the Android 13 platform and the notification drawers when the user selects “Allow” or “Don’t allow” are shown below.

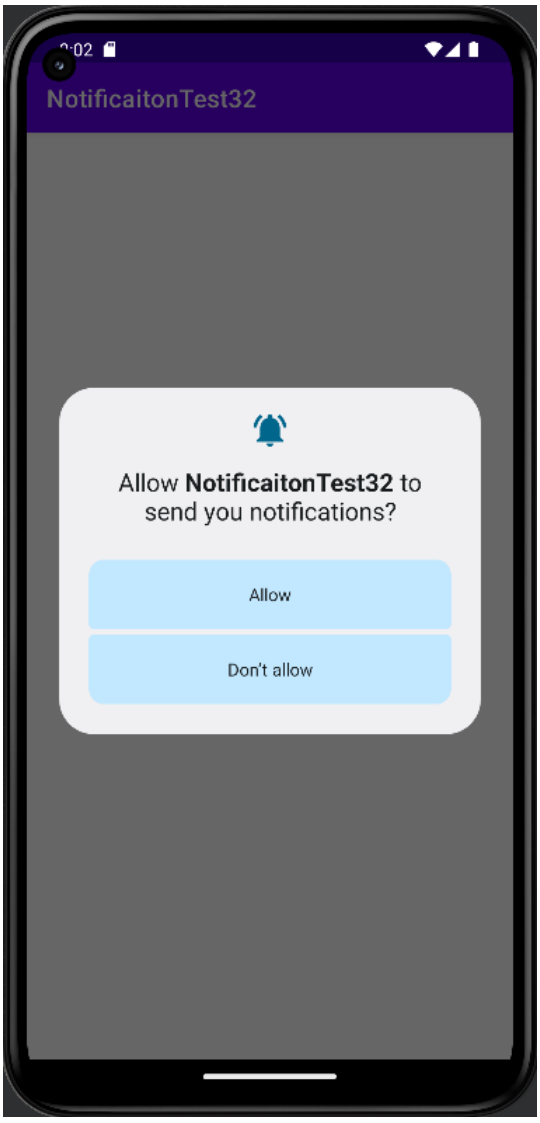


Fig. 4.10.4: Permissions Dialog

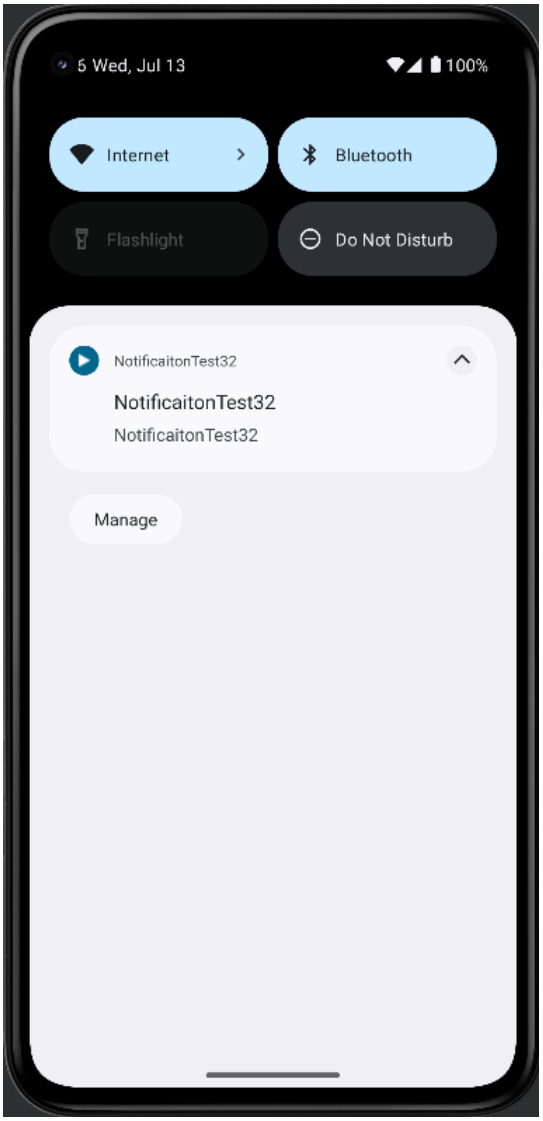


Fig. 4.10.5: Notification Drawer “Allow”

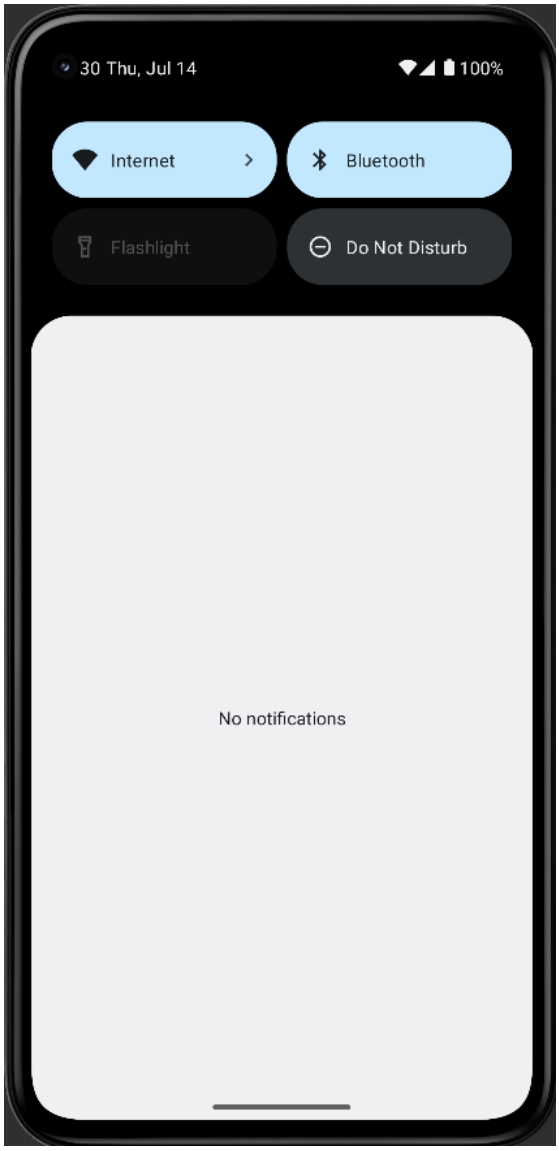


Fig. 4.10.6: Notification Drawer “Don’t Allow”

For devices running targetSdkVersion 33 and above:

If the above code is changed to targetSdkVersion 33 or higher and executed on the Android 13 platform in the same way, the permission dialog will not appear and notification will not appear in the notification drawer. The ON/OFF setting of the notification feature itself is disabled when checked from the app-specific settings.

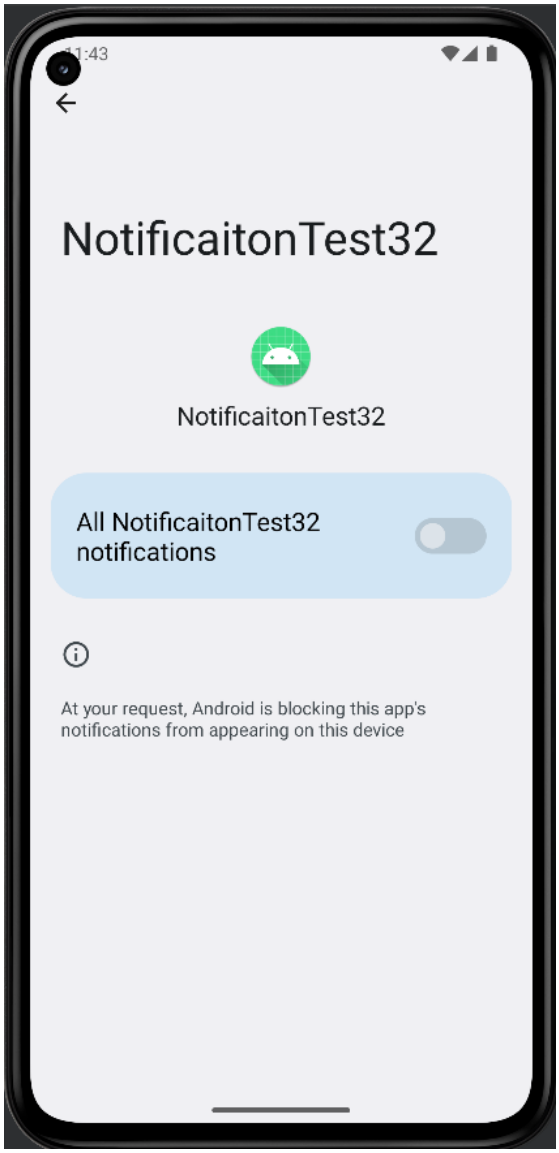


Fig. 4.10.7: ON/OFF Setting of Notification Feature Itself Is Disabled

This is because Android 13 (API level 33) introduced POST_NOTIFICATIONS, a new runtime permission to send notifications. When building with targetSdkVersion 33 or higher, it is necessary to declare POST_NOTIFICATIONS in the manifest file and implement displaying of a separate permission dialog.

The following is an example of declaration in the manifest file.

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    package="xxx.xxxx.myapplication">

<uses-permission android:name="android.permission.POST_NOTIFICATIONS"/>
```

Also, the following is an example of implementation displaying the permission dialog.

```
private ActivityResultLauncher<String> requestPermissionLauncher =
        registerForActivityResult(new ActivityResultContracts.RequestPermission(),␣
↪isGranted -> {
            if (isGranted) {
                // Permission is granted. Continue the action or workflow in your
```

(continues on next page)

(continued from previous page)

```
                // app.
            } else {
                // Explain to the user that the feature is unavailable because the
                // features requires a permission that the user has denied. At the
                // same time, respect the user's decision. Don't link to system
                // settings in an effort to convince the user to change their
                // decision.
            }
            });

@Override
protected void onCreate(Bundle savedInstanceState) {
    // ...
    buttonStart.setOnClickListener( v -> {
        // ...
        requestPermissionLauncher.launch(Manifest.permission.POST_NOTIFICATIONS);
        startForegroundService(intent);
    });
```

The following shows the permission dialog when the above code is executed on the Android 13 platform.

Fig. 4.10.8: Permission Dialog

It is recommended that these changes be addressed “as soon as possible” when targetSdkVersion 33 or higher is used.

Here is a quote from the official text[36].

“We highly recommend that you target Android 13 or higher as soon as possible to benefit from the additional control and flexibility of this feature. If you continue to target 12L (API level 32) or lower, you lose some flexibility with requesting the permission in the context of your app’s functionality.”

4.10.3.4 Change in Operation for Notifications Indicating Progress

Notifications that indicate that a process is in progress, such as music media being played or downloaded, can be set to remain in the notification drawer by setting Notification.Builder#setOngoing(true) and setting FLAG_ONGOING_EVENT.

```
NotificationCompat.Builder builder
        = new NotificationCompat.Builder(this, "CHANNEL_ID")
        .setSmallIcon(android.R.drawable.ic_menu_info_details)
        .setContentTitle("Notification Title")
        .setContentText("Notification Message")
        .setPriority(NotificationCompat.PRIORITY_DEFAULT
        );

builder.setOngoing(true);
```

This way, the user was prevented from accidentally closing the notification. However, this behavior has changed since Android 14, allowing users to close notifications even if FLAG_ONGOING_EVENT is set.

The differences in behavior when the above code is executed on Android 11 and Android 14 are shown below.

First, this shows the behavior on Android 11.

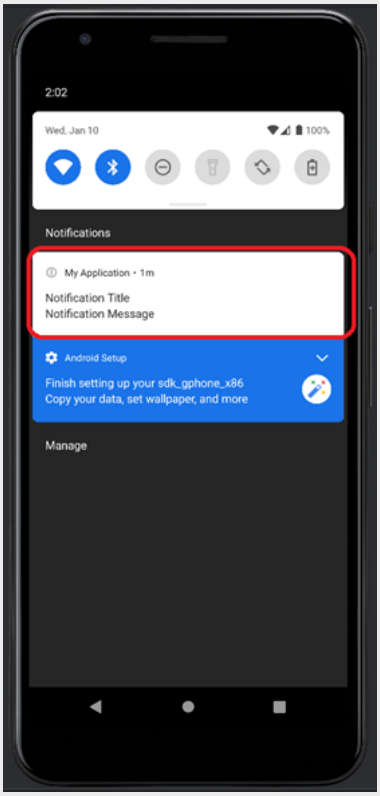


Fig. 4.10.9: Android 11 Notification

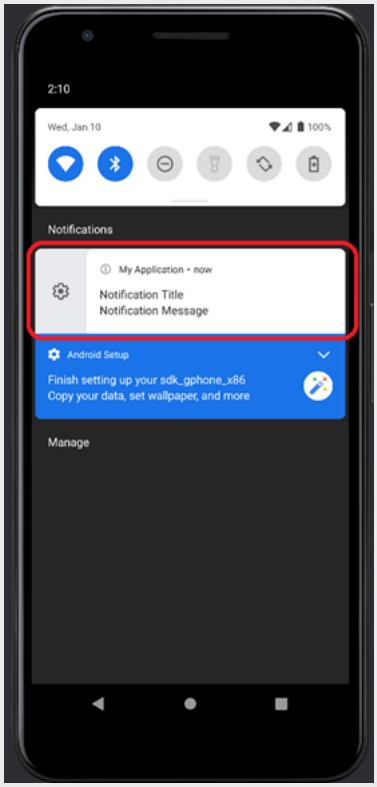


Fig. 4.10.10: Swiping to the right does not remove the notification

[36] https://developer.android.com/about/versions/13/changes/notification-permission

Next, this shows the behavior on Android 14.

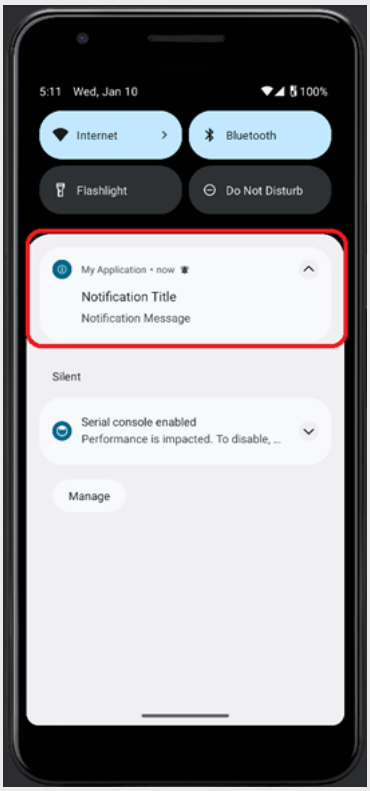


Fig. 4.10.11: Android 14 Notification

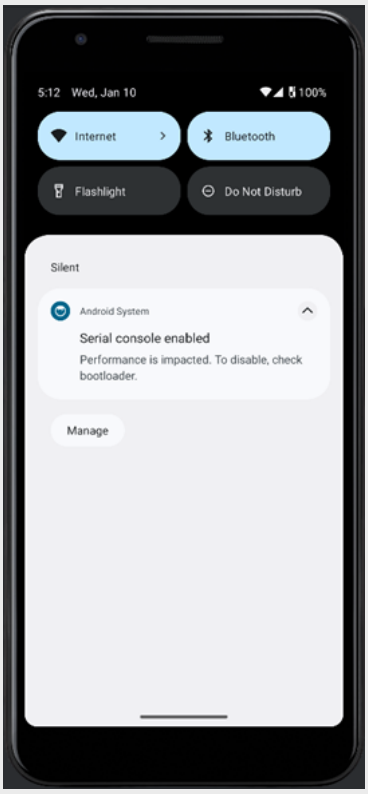


Fig. 4.10.12: Swiping to the right removes the notification

Nonetheless, in the following situations, the above behavior does not apply, and the notification will continue to remain in the notification drawer.

- When the smartphone is locked
- When the user selects the “Clear All” notification action (to prevent accidental deletion)

The above behavior also does not apply to the following use cases.

- Notifications created using MediaStyle
- When policy restricts use to security and privacy cases
- Device Policy Controller (DPC) and support packages for enterprises

4.11 Using Shared Memory

Previously, the Android OS included a shared memory mechanism, and it was provided by android.os.MemoryFile. However, it did not directly provide APIs or access control for sharing over multiple applications, and it was difficult to use for general applications. In Android 8.1 (API level 27), the android.os.SharedMemory package was introduced, which enabled the shared memory mechanism to be used relatively easily from general applications. At the time of Android 8.1, MemoryFile is a wrapper of SharedMemory, and use of SharedMemory is recommended. This section describes the important security points when using this SharedMemory API.

As described later, this API was built assuming a structure where a provided application and memory are shared when a service of an application creates a shared memory and provides this shared memory to other applications. And so, all the information described in "4.4. *Creating/Using Services*" also applies to applications that provide shared memory and applications that use this shared memory. If you have not already read this information, it is recommended that you read "4.4. *Creating/Using Services*" before proceeding to the explanation below.

No Android Support Library is available that supports the SharedMemory API. For this reason, to operate applications using SharedMemory in systems that are below API level 27, measures are required such as by implementing an equivalent virtual memory mechanism, such as by wrapping C language level APIs using JNI, and the processes must be separated based on the version.

4.11.1 Overview of Android Shared Memory

Shared memory is a mechanism for sharing the same physical memory area among multiple applications.

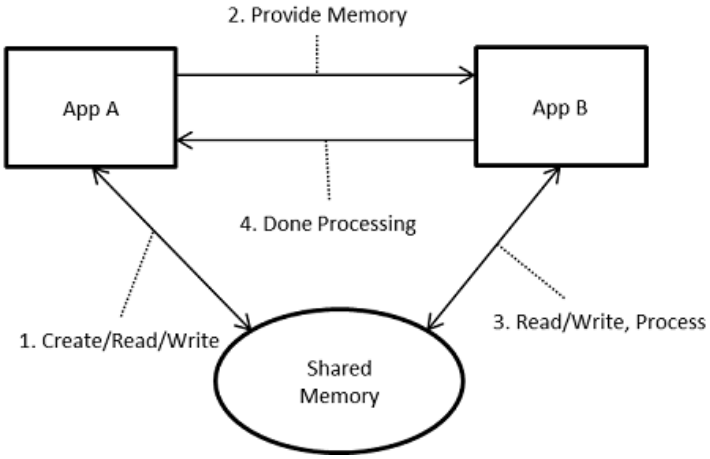


Fig. 4.11.1: Overview of Shared Memory

The figure above shows the appearance when using a shared memory for application A and application B. Application A creates a shared memory object, and it is provided to application B. The role of providing shared memory by application A is handled as a service of application A. Application B connects to this service, requests and obtains the shared memory, and after the processes required by the shared memory are completed, application B notifies application A that use is completed.

For example, if handling data where the maximum size (1 MB[37]) for allowable communication between normal processes is exceeded, such as bitmap data of a large image, shared memory can be used to enable sharing among multiple processes. Also, the amount of memory used for the entire device can be reduced for enabling normal memory access, and this allows for extremely high-speed communication between processes. However, because multiple applications are simultaneously accessing in parallel, consideration must also be made for maintaining the integrity of the data in certain cases. To avoid this, exclusive control can be performed between applications, and other careful designs are needed to ensure that the memory area is properly divided and the accessed areas do not interfere with each other.

As mentioned above, the shared memory API of Android SDK was built so that a service creates a shared memory object and provides it to other processes. Because the shared memory class (android.os.SharedMemory) is defined as parcelable, the shared memory instance can be easily passed on to other processes through binders. An overview of the exchanges between the service and client in the sample code appearing later has the structure shown in the figure below (this can vary significantly depending on the structure of the service).

[37] https://developer.android.com/guide/components/activities/parcelables-and-bundles

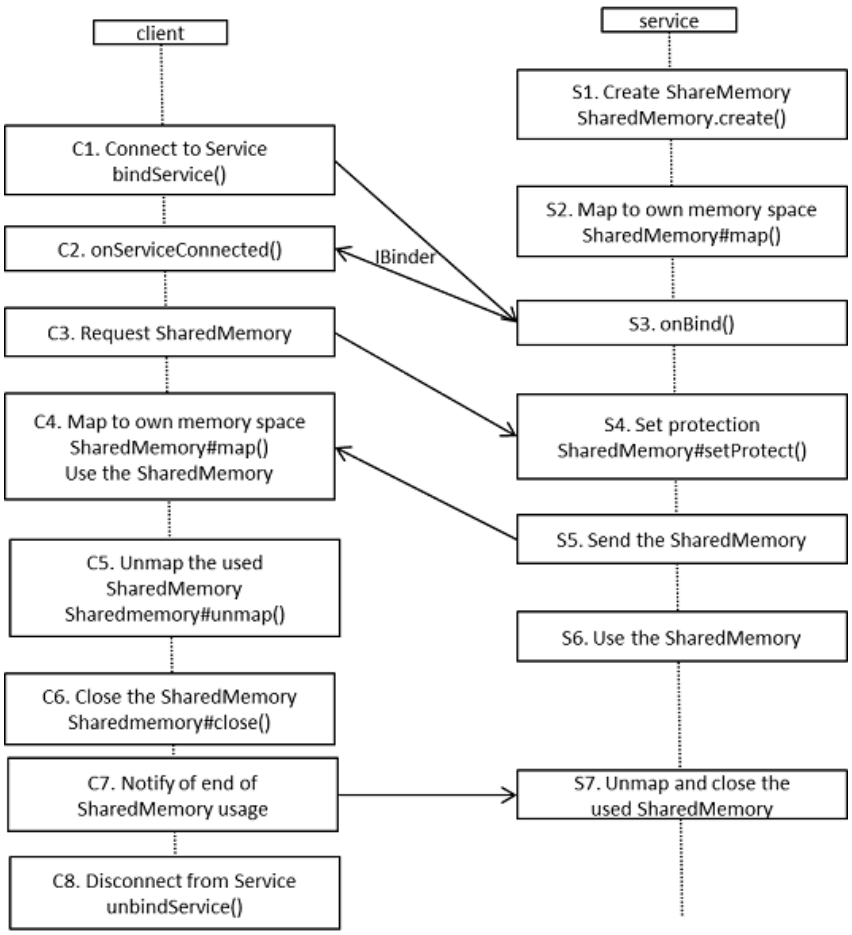


Fig. 4.11.2: Exchanges between shared memory service and client

- S1. A service uses SharedMemory.create() to create a shared memory.
- S2. If the service itself will use the shared memory, SharedMemory#map() is used to map the shared memory to its own memory space.
- C1. The client uses explicit intent to connect to the service by Context#bindService().
- S3. When a connection request is received from the client, the service's onBind() call back is called. The service performs the required pre-processing (if needed) at this stage and returns a IBinder for connection to the client.
- C2. The return value (IBinder instance) when the service executed onBind() is returned as an argument of onServiceConnected() callback on the client side. Then, this IBinder is used to perform communication with the service.
- C3. The client requests the shared memory for the service.
- S4. The service receives a shared memory request from the client and sets the operations permitted (read, write) when the client accesses the shared memory.
- S5. The service passes on the shared memory object to the client.
- C4. To access the received shared memory, the client maps the shared memory to its own address space for use.
- C5, C6. When the client has finished use of the shared memory, the shared memory is unmapped (C5) from its own memory space, and the shared memory is closed (C6).
- C7. Then, the client notifies the server that use of the shared memory is completed.
- C8. The client disconnects from the service.

- S7. After the message that usage is completed is received from the client, the service itself also unmaps and closes the shared memory.

The onServiceConnected() in item C2 above is defined as a class where the android.content.ServiceConnection class is implemented. For specific examples, refer to the sample code appearing later. Several communication methods using IBinder are available, but Messenger is used in the sample code.

4.11.2 Sample Code

As described before, the side that creates the shared memory and provides it to other applications is implemented as a service. For this reason, from the standpoint of security for functions and information sharing, there are no fundamental differences from the information contained in "4.4. *Creating/Using Services*" Based on the classifications in 4.4., the figure below shows the process for determining who the memory will be shared with.

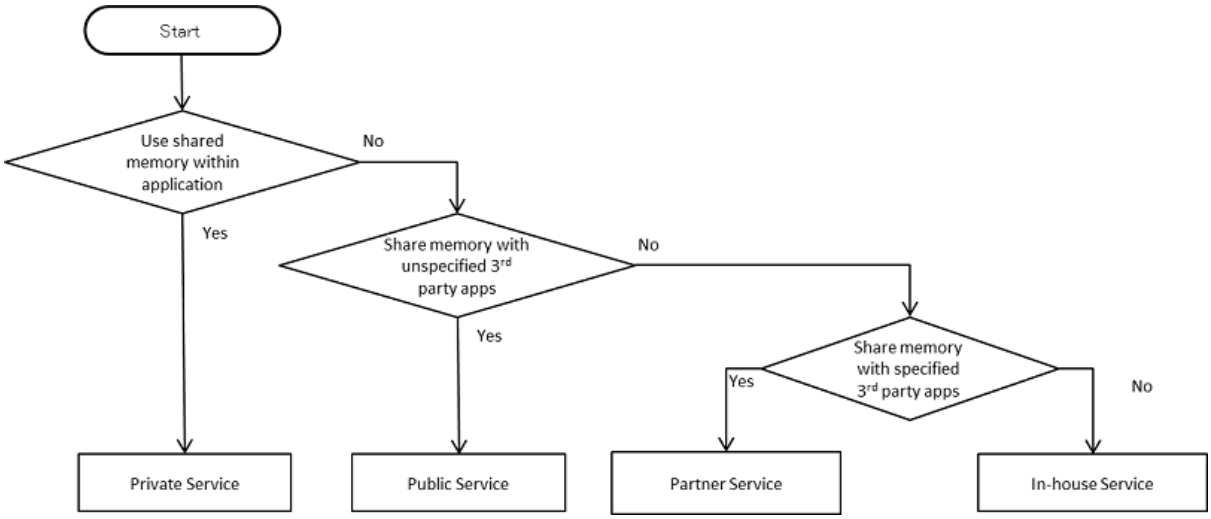


Fig. 4.11.3: Flow Figure to select SharedMemory Service Type

Table 4.4.2 in "4.4.1. *Sample Code*" describes how a service is implemented, but for shared memory, sharing with other applications must be implemented using a binder. And so, shared memory cannot be implemented as a start-Service or IntentService service. For this reason, it is implemented as shown in the table below.

Table 4.11.1: Service Category and Types(Shared Memory)

Category	Private Service	Public Service	Partner Service	In-house Service
startService type	-	-	-	-
IntentService type	-	-	-	-
local bind type	OK	-	-	-
Messenger bind type	OK	OK	-	**OK***
AIDL bind type	OK	OK	OK	OK

The overall structure is virtually identical to that in "4.4.1. *Sample Code*" Also, because the items specific to shared memory are the same in all cases, in the specific sample code, the items marked with an asterisk in the above table indicate those that apply to in-house services only. For this reason, to use shared memory in other cases, refer to the information from "4.4.1.1. *Creating/Using Private Services*" to "4.4.1.3. *Creating/Using Partner Services*".

4.11.2.1 Creating/Using Private Services

In this case, a structure is used that shares shared memory created by a private service between multiple processes contained in the application. Also, this private service is started as a process independent from the main process of the application.

Points:

1. The service that creates the shared memory is explicitly set to private by exported=”false”.
2. If a process in an application references data that was written by another process, the safety is verified even if it is a process within the same application.
3. Sensitive information can be shared because the sharing of memory is a process within the same application.

The sample code in "4.4.1.1. *Creating/Using Private Services*" used services by Intent, but for shared memory, memory resources cannot be shared through Intent, and so a method based on local bind, Message bind, or AIDL bind must be used.

4.11.2.2 Creating/Using Public Services

As described in "4.4.1.2. *Creating/Using Public Services*," a public service is a service which is assumed to be used by an unspecified large number of applications. As a result, use by malware must also be assumed. Generally, attention must be paid to the points mentioned in 4.4.1.2., but those points are rephrased below from the standpoint of shared memory.

Points (Creating a Service):

1. Explicitly set to public using exported=”true”.
2. Verify the safety of parameters and data contained in requests and other operations for starting services and sharing memory.
3. Sensitive information must not be shared using shared memory.

Points (Using a Service): 1. Sensitive information must not be written to shared memory. 2. Safety is verified when referencing data that was written by another application.

4.11.2.3 Creating/Using Partner Services

This information is virtually identical to the information shown in "4.4.1.3. *Creating/Using Partner Services*", but this is rephrased from the standpoint of shared memory for showing the following points (Like the sample code in 4.4.1.3., this assumes use of the AIDL bind service)

Points (Creating a Service):

1. Do not define the Intent Filter, and explicitly declare exported=”true”.
2. Verify the requesting application’s certificate through a predefined whitelist.
3. onBind(onStartCommand, onHandleIntent) cannot be used to determine whether the requester is a partner.
4. Verify the safety of received Intent even if the Intent was sent from a partner application.
5. Writing to the shared memory is permissible only for information that is allowed to be disclosed to the partner application.

Points (Using a Service):

1. Verify that the certificate of the requesting partner service application is registered in the whitelist.
2. Writing to the shared memory is permissible only for information that is allowed to be disclosed to the requesting partner application.
3. Use explicit Intent to call a partner service.
4. Verify the safety of the data even if the data was written by a partner application.

4.11.2.4 Creating/Using In-house Services

This section presents an example where shared memory is provided by a service available as public, but the shared memory is provided to an in-house application only. Like the example in "4.4.1.4. *Creating/Using In-house Services*", a Messenger bind service is used. The principles and settings for the background are described in 4.4.1.4., and so refer to 4.4.1.4. first if you have not already read this information.

Sample code for application at service side (Messenger bind)

Points are shown below, but items 1 to 5 and 7 are presented in "4.4.1.4. *Creating/Using In-house Services*," and item 6 is the only item specific to shared memory.

Points:

1. Define an in-house signature permission.
2. Request declaration of the in-house signature permission.
3. Do not define the Intent Filter, and explicitly declare exported="true".
4. Verify that the in-house signature permission is defined by an in-house application.
5. Verify the safety of received Intent even if the Intent was sent from an in-house application.
6. Before passing the shared memory on to a client, use SharedMemory#setProtect() to limit the available operations by the client.
7. Sign the APK using the same developer key as the requesting application.

For purposes of simplification, this example defines the service that allocates the shared memory and the activity that uses the service within the same application (service is started as a separate process within the same application). For this reason, both the signature permission definition and use declaration are contained in the manifest file.

```
AndroidManifest.xml
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
          >

  <!-- *** POINT 1 *** Define an in-house signature permission -->
  <permission android:name="org.jssec.android.sharedmemory.inhouseservice.
↪messenger.MY_PERMISSION"
              android:protectionLevel="signature" />

  <!-- *** POINT 8 *** Define an in-house signature permission -->
  <uses-permission
      android:name="org.jssec.android.service.inhouseservice.messenger.MY_
↪PERMISSION" />

  <application
      android:allowBackup="false"
      android:icon="@mipmap/ic_launcher"
      android:label="@string/app_name"
      android:roundIcon="@mipmap/ic_launcher_round"
      android:supportsRtl="true"
      android:theme="@style/AppTheme">
    <activity android:name="org.jssec.android.sharedmemory.inhouseservice.
↪messenger.MainActivity"
        android:exported="true">
      <intent-filter>
        <action android:name="android.intent.action.MAIN" />

        <category android:name="android.intent.category.LAUNCHER" />
      </intent-filter>
```

(continues on next page)

(continued from previous page)

```
    </activity>
    <!-- Service which utilizes Messenger -->
    <!-- *** POINT 2 *** Request declaration of the in-house signature permission -
↪->
    <!-- *** POINT 3 *** Do not define the Intent Filter, and explicitly declare␣
↪exported="true" -->
    <!-- For purposes of simplification, make the service which provide shared␣
↪memory to be a different process in the same application -->
    <service android:name="org.jssec.android.sharedmemory.inhouseservice.messenger.
↪SHMService"
             android:exported="true"
             android:permission="org.jssec.android.sharedmemory.inhouseservice.
↪messenger.MY_PERMISSION"
             android:process=".shmService" />
  </application>

</manifest>
```

SHMService.java

```
/*
 * Copyright (C) 2012-2026 Japan Smartphone Security Association
 *
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 *
 *      http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */

package org.jssec.android.sharedmemory.inhouseservice.messenger;

import org.jssec.android.shared.SigPerm;
import org.jssec.android.shared.Utils;

import android.app.Service;
import android.content.Context;
import android.content.Intent;
import android.content.pm.PackageManager;
import android.os.Handler;
import android.os.IBinder;
import android.os.Message;
import android.os.Messenger;
import android.os.RemoteException;
import android.os.SharedMemory;
import android.system.ErrnoException;
import android.util.Log;
import android.widget.Toast;

import java.nio.ByteBuffer;
```

(continues on next page)

(continued from previous page)

```
import static android.content.pm.PackageManager.PERMISSION_GRANTED;
import static android.system.OsConstants.PROT_EXEC;
import static android.system.OsConstants.PROT_READ;
import static android.system.OsConstants.PROT_WRITE;

public class SHMService extends Service {
    // In-house Signature Permission
    private static final String MY_PERMISSION =
        "org.jssec.android.sharedmemory.inhouseservice.messenger.MY_PERMISSION";

    // Hash value of the certificate of In-house applications
    private static String sMyCertHash = null;
    private static String myCertHash(Context context) {
        if (sMyCertHash == null) {
            if (Utils.isDebuggable(context)) {
                // Hash value of the certificate "androiddebugkey" stored in
                // debug.keystore
                sMyCertHash = "0EFB7236 328348A9 89718BAD DF57F544 D5CCB4AE␣
↪B9DB34BC 1E29DD26 F77C8255";
            } else {
                // Hash value of the certificate "my company key" in keystore
                sMyCertHash = "D397D343 A5CBC10F 4EDDEB7C A10062DE 5690984F␣
↪1FB9E88B D7B3A7C2 42E142CA";
            }
        }
        return sMyCertHash;
    }

    private final String TAG = "SHM";

    // Strings which will be sent to client
    private final String greeting = "Hi! I send you my memory. Let's Share it!";
    // Page size is 4K bytes
    public static final int PAGE_SIZE = 1024 * 4;
    // In this example, we use two SharedMemory objects
    // Client side specify the one of these SharedMemory by using following
    // identify
    public static final int SHMEM1 = 0;
    public static final int SHMEM2 = 1;

    // Instances of Shared Memory
    // mSHMem1: used for sending data to client
    private SharedMemory mSHMem1 = null;
    // ByteBuffer for mapping mSHMem
    private ByteBuffer m1Buffer1;
    // mSHMem2: used for receiving data from client side
    private SharedMemory mSHMem2 = null;
    // ByteBuffer for mapping mSHMem2
    private ByteBuffer m2Buffer1;
    private ByteBuffer m2Buffer2;
    // true iff all ByteBuffers are mapped successfully
    private boolean mBufferMapped = false;


    // In this example, Messenger is used for communicating with client
    // The follwings are message identifier for the communication
```

(continues on next page)

(continued from previous page)

```
    public static final int MSG_INVALID = Integer.MIN_VALUE;
    public static final int MSG_ATTACH  =
            MSG_INVALID + 1; // client requests SHMEM1
    public static final int MSG_ATTACH2 =
            MSG_ATTACH + 1;  // client requests SHMEM2
    public static final int MSG_DETACH  =
            MSG_ATTACH2 + 1; // client no more need SHMEM1
    public static final int MSG_DETACH2 =
            MSG_DETACH + 1;  // client no more need SHMEM2
    public static final int MSG_REPLY1  =
            MSG_DETACH2 + 1; // first reply from client
    public static final int MSG_REPLY2  =
            MSG_REPLY1 + 1;  // second reply from client
    public static final int MSG_END     =
            MSG_REPLY2 + 1;  // Service declared the end of the session

    // Handler manipulating Message received from client
    private class CommHandler extends Handler {
        @Override
        public void handleMessage(Message msg) {
            switch (msg.what) {
            case MSG_ATTACH:
                Log.d(TAG, "got MSG_ATTACH");
                shareWith1(msg);
                break;
            case MSG_ATTACH2:
                Log.d(TAG, "got MSG_ATTACH2");
                shareWith2(msg);
                break;
            case MSG_DETACH:
                Log.d(TAG, "got MSG_DETACH");
                unShare(msg);
                break;
            case MSG_REPLY1:
                Log.d(TAG, "got MSG_REPLY1");
                gotReply(msg);
                break;
            case MSG_REPLY2:
                Log.d(TAG, "got MSG_REPLY2");
                gotReply2(msg);
                break;
            default:
                invalidMsg(msg);
            }
        }
    }

    private final Handler mHandler = new CommHandler();

    // Messenger used for receiving data from client
    private final Messenger mMessenger = new Messenger(mHandler);

    // When bound, extract Binfer from Message, pass it to client
    @Override
    public IBinder onBind(Intent intent) {
        // ** POINT 4 *** Verify that the in-house signature permission is defined
```

(continues on next page)

(continued from previous page)

```
        // by an in-house application.
        if (!SigPerm.test(this, MY_PERMISSION, myCertHash(this))) {
            Toast.makeText(this, "In-house signature permission is not defined by␣
↪an in-house application.", Toast.LENGTH_LONG).show();
            return null;
        }

        // *** POINT 5 *** Verify the safety of received Intent even if the Intent
        // was sent from an in-house application
        // Omitted because this is an sample code. Refer to
        // "3.2 Handling Input Data Carefully and Securely".
        String param = intent.getStringExtra("PARAM");
        Log.d(TAG, String.format("Received Parameter [%s]!", param));
        return mMessenger.getBinder();
    }

    // Mapping layout
    // Offset must be page boundary
    public static final int SHMEM1_BUF1_OFFSET = 0;
    public static final int SHMEM1_BUF1_LENGTH = 1024;
    public static final int SHMEM2_BUF1_OFFSET = 0;
    public static final int SHMEM2_BUF1_LENGTH = 1024;
    public static final int SHMEM2_BUF2_OFFSET = PAGE_SIZE;
    public static final int SHMEM2_BUF2_LENGTH = 128;

    // Allocate 2 SharedMemory objects
    private boolean allocateSharedMemory() {
        try {
            // For sending data to client
            mSHMem1 = SharedMemory.create("SHM", PAGE_SIZE);
            // For receiving data from client
            mSHMem2 = SharedMemory.create("SHM2", PAGE_SIZE * 2);
        } catch (ErrnoException e) {
            Log.e(TAG, "failed to allocate shared memory" + e.getMessage());
            return false;
        }
        return true;
    }

    // Map specified SharedMemory
    private ByteBuffer mapShared(SharedMemory mem,
                                 int prot, int offset, int size) {
        ByteBuffer tBuf ;
        try {
            tBuf = mem.map(prot, offset, size);
        } catch (ErrnoException e) {
            Log.e(TAG, "could not map, prot=" + prot + ", offset=" + offset + ",␣
↪length=" + size + "\n " + e.getMessage() + "err no. = " + e.errno);
            return null;
        } catch (IllegalArgumentException e){
            Log.e(TAG, "map failed: " + e.getMessage());
            return null;
        }
        Log.d(TAG, "mmap success: prot=" + prot);
        return tBuf;
    }
```

(continues on next page)

(continued from previous page)

```
    // Server side mappings of SharedMemory objects
    private void mapMemory() {
        // mSHMem1: read/write
        m1Buffer1 = mapShared(mSHMem1,
            PROT_READ | PROT_WRITE | PROT_EXEC, SHMEM1_BUF1_OFFSET,
            SHMEM1_BUF1_LENGTH);
        // mSHMem2: separate two regions, read/write for each
        m2Buffer1 = mapShared(mSHMem2,
            PROT_READ | PROT_WRITE, SHMEM2_BUF1_OFFSET,
            SHMEM2_BUF1_LENGTH);
        m2Buffer2 = mapShared(mSHMem2,
            PROT_READ | PROT_WRITE, SHMEM2_BUF2_OFFSET,
            SHMEM2_BUF2_LENGTH);

        if (m1Buffer1 != null && m2Buffer1 != null && m2Buffer2 != null) {
            mBufferMapped = true;
        }
    }

    // Free SharedMemory
    private void deAllocateSharedMemory () {
        if (mBufferMapped) {
            if (mSHMem1 != null) {
                if (m1Buffer1 != null) SharedMemory.unmap(m1Buffer1);
                m1Buffer1 = null;
                mSHMem1.close();
                mSHMem1 = null;
            }
            if (mSHMem2 != null) {
                if (m2Buffer1 != null) SharedMemory.unmap(m2Buffer1);
                if (m2Buffer2 != null) SharedMemory.unmap(m2Buffer2);
                m2Buffer1 = null;
                m2Buffer2 = null;
                mSHMem2.close();
                mSHMem2 = null;
            }
            mBufferMapped = false;
        }
    }

    @Override
    public void onCreate() {
        super.onCreate();

        // Allocate SharedMemory objects at the time of instantiation
        // If succeded, map SharedMemory objects
        if (allocateSharedMemory()) {
            mapMemory();
        }
    }

    // Provide SHMEM1 to client
    private void shareWith1(Message msg){
        // If failed in allocating or mapping, do nothing
        if (!mBufferMapped) return;
```

(continues on next page)

(continued from previous page)

```
        // *** POINT 6 *** Before passing the shared memory on to a client, use
        // SharedMemory#setProtect() to limit the available operations by the
        // client.
        // Client can only read from mSHMem1
        mSHMem1.setProtect(PROT_READ);

        // Mapping hash been done before the above setProtect(PROT_READ),
        // so setver side can write to mShMem1 via m1Buffer1
        // Put the size of the messege, then add message string
        m1Buffer1.putInt(greeting.length());
        m1Buffer1.put(greeting.getBytes());

        try {
            // Pass the SharedMemory object to the client
            Message sMsg = Message.obtain(null, SHMEM1, mSHMem1);
            msg.replyTo.send(sMsg);
        } catch (RemoteException e) {
            Log.e(TAG, "Failed to share" + e.getMessage());
        }
    }

    // Provide SHMEM2
    private void shareWith2(Message msg) {
        if (!mBufferMapped) return;

        // *** POINT 6 *** Before passing the shared memory on to a client, use
        // SharedMemory#setProtect() to limit the available operations by the
        // client.
        // Client can write to mSHMem2
        mSHMem2.setProtect(PROT_WRITE);
        // Set messages to client in each buffer
        final String greeting2 = "You can write here!";
        m2Buffer1.putInt(greeting2.length());
        m2Buffer1.put(greeting2.getBytes());
        final String greeting3 = "From this point, I'll also write.";
        m2Buffer2.putInt(greeting3.length());
        m2Buffer2.put(greeting3.getBytes());
        try {
            // Pass the shared memory objects to the client
            Message sMsg = Message.obtain(null, SHMEM2, mSHMem2);
            msg.replyTo.send(sMsg);
        } catch (RemoteException e){
            Log.e(TAG, "failed to share mSHMem2" + e.getMessage());
        }
    }

    // Stop sharing memory
    private void unShare(Message msg){
        deAllocateSharedMemory();
    }

    // Accepted invalid message
    private void invalidMsg(Message msg){
        Log.e(TAG, "Got an Invalid message: " + msg.what);
    }
```

(continues on next page)

(continued from previous page)

```
    // Retrive data which set by the client from buffer
    // The first element is a size of the data followed by byte sequence of a
    // string
    private String extractReply (ByteBuffer buf){
        int len = buf.getInt();
        byte [] bytes = new byte[len];
        buf.get(bytes);
        return new String(bytes);
    }

    // In this example, server side accepts two types of message from the client
    // goReply() assumes that  m1Buffer1 holds a data from the client
    private void gotReply(Message msg) {
        m1Buffer1.rewind();
        String message = extractReply(m1Buffer1);
        if (!message.equals(greeting)){
            Log.e(TAG, "my message was overwritten: " + message);
        }
    }
    // got Reply2() assumes m2Buffer1 holds a data from the client
    private void gotReply2(Message msg) {
        m2Buffer1.rewind();
        String message = extractReply(m2Buffer1);
        android.util.Log.d(TAG, "got a message of length " + message.length() +
                                " from client: " + message);
        // Accepting a message in m2Buffer1 is a sign of the end of sharing memory
        Message eMsg = Message.obtain();
        eMsg.what = MSG_END;
        try {
            msg.replyTo.send(eMsg);
        } catch (RemoteException e){
            Log.e(TAG, "error in reply 2: " + e.getMessage());
        }
    }
}
```

SigPerm.java

```
/*
 * Copyright (C) 2012-2026 Japan Smartphone Security Association
 *
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 *
 *      http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */

package org.jssec.android.shared;
```

(continues on next page)

(continued from previous page)

```
import android.content.Context;
import android.content.pm.PackageManager;
import android.content.pm.PackageManager.NameNotFoundException;
import android.content.pm.PermissionInfo;
import android.os.Build;

import static android.content.pm.PackageManager.CERT_INPUT_SHA256;

public class SigPerm {

    public static boolean test(Context ctx, String sigPermName,
                               String correctHash) {
        if (correctHash == null) return false;
        correctHash = correctHash.replaceAll(" ", "");
        try {
            // Get the package name of the application which declares a permission
            // named sigPermName.
            PackageManager pm = ctx.getPackageManager();
            PermissionInfo pi =
                pm.getPermissionInfo(sigPermName, PackageManager.GET_META_DATA);
            String pkgname = pi.packageName;
            // Fail if the permission named sigPermName is not a Signature
            // Permission
            if (pi.protectionLevel != PermissionInfo.PROTECTION_SIGNATURE)
                return false;

            // Compare the actual hash value of pkgname with the correct hash
            // value.
            if (Build.VERSION.SDK_INT >= 28) {
                // ** if API Level >= 28, direct check is possible
                return pm.hasSigningCertificate(pkgname,
                                                Utils.hex2Bytes(correctHash),
                                                CERT_INPUT_SHA256);
            } else {
                // else(API Level < 28) use the facility of PkgCert
                return correctHash.equals(PkgCert.hash(ctx, pkgname));
            }

        } catch (NameNotFoundException e) {
            return false;
        }
    }
}
```

PkgCert.java

```
/*
 * Copyright (C) 2012-2026 Japan Smartphone Security Association
 *
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 *
 *      http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
```

(continues on next page)

(continued from previous page)

```
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */

package org.jssec.android.shared;

import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;

import android.content.Context;
import android.content.pm.PackageInfo;
import android.content.pm.PackageManager;
import android.content.pm.PackageManager.NameNotFoundException;
import android.content.pm.Signature;

public class PkgCert {

    public static boolean test(Context ctx, String pkgname, String correctHash) {
        if (correctHash == null) return false;
        correctHash = correctHash.replaceAll(" ", "");
        return correctHash.equals(hash(ctx, pkgname));
    }

    public static String hash(Context ctx, String pkgname) {
        if (pkgname == null) return null;
        try {
            PackageManager pm = ctx.getPackageManager();
            PackageInfo pkginfo =
                pm.getPackageInfo(pkgname, PackageManager.GET_SIGNATURES);
            // Will not handle multiple signatures.
            if (pkginfo.signatures.length != 1) return null;
            Signature sig = pkginfo.signatures[0];
            byte[] cert = sig.toByteArray();
            byte[] sha256 = computeSha256(cert);
            return byte2hex(sha256);
        } catch (NameNotFoundException e) {
            return null;
        }
    }

    private static byte[] computeSha256(byte[] data) {
        try {
            return MessageDigest.getInstance("SHA-256").digest(data);
        } catch (NoSuchAlgorithmException e) {
            return null;
        }
    }

    private static String byte2hex(byte[] data) {
        if (data == null) return null;
        final StringBuilder hexadecimal = new StringBuilder();
        for (final byte b : data) {
            hexadecimal.append(String.format("%02X", b));
        }
        return hexadecimal.toString();
```

(continues on next page)

(continued from previous page)

```
    }
}
```

*** Point 7 *** When exporting an APK, sign the APK with the same developer key as the requesting application.

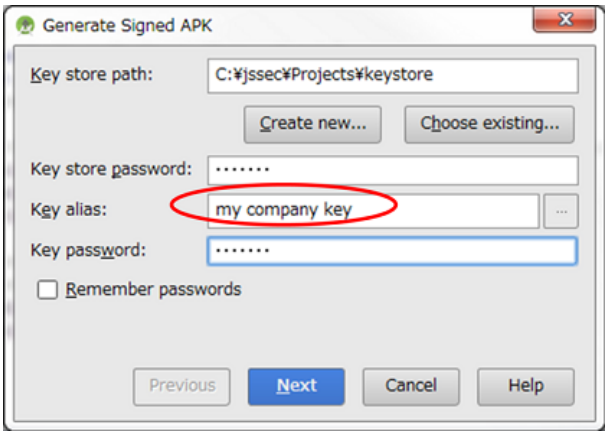


Fig. 4.11.4: Signing the APK with the same developer key as the requesting application

Sample code for client

Points:

8. Declare use of the in-house signature permission.
9. Verify that the in-house-defined signature permission is defined by the in-house application.
10. Verify that the destination application is signed by the in-house certificate.
11. Sensitive information can be sent because the destination application is in-house.
12. Use explicit Intent to call an in-house service.
13. Sign the APK using the same developer key as the destination application.

All the points shown here are the same as the points for the client in "4.4.1.4. *Creating/Using In-house Services*", and no points are specific to shared memory. Basic points on using shared memory are shown in the sample code below, and so refer to it for further information.

```
AndroidManifest.xml
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
          >

  <!-- *** POINT 1 *** Define an in-house signature permission -->
  <permission android:name="org.jssec.android.sharedmemory.inhouseservice.
↪messenger.MY_PERMISSION"
              android:protectionLevel="signature" />

  <!-- *** POINT 8 *** Define an in-house signature permission -->
  <uses-permission
      android:name="org.jssec.android.service.inhouseservice.messenger.MY_
↪PERMISSION" />

  <application
      android:allowBackup="false"
      android:icon="@mipmap/ic_launcher"
```

(continues on next page)

(continued from previous page)

```
      android:label="@string/app_name"
      android:roundIcon="@mipmap/ic_launcher_round"
      android:supportsRtl="true"
      android:theme="@style/AppTheme">
    <activity android:name="org.jssec.android.sharedmemory.inhouseservice.
↪messenger.MainActivity"
        android:exported="true">
      <intent-filter>
        <action android:name="android.intent.action.MAIN" />

        <category android:name="android.intent.category.LAUNCHER" />
      </intent-filter>
    </activity>
    <!-- Service which utilizes Messenger -->
    <!-- *** POINT 2 *** Request declaration of the in-house signature permission -
↪->
    <!-- *** POINT 3 *** Do not define the Intent Filter, and explicitly declare␣
↪exported="true" -->
    <!-- For purposes of simplification, make the service which provide shared␣
↪memory to be a different process in the same application -->
    <service android:name="org.jssec.android.sharedmemory.inhouseservice.messenger.
↪SHMService"
             android:exported="true"
             android:permission="org.jssec.android.sharedmemory.inhouseservice.
↪messenger.MY_PERMISSION"
             android:process=".shmService" />
  </application>

</manifest>
```

MainActivity.java

```
/*
 * Copyright (C) 2012-2026 Japan Smartphone Security Association
 *
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 *
 *      http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */

package org.jssec.android.sharedmemory.inhouseservice.messenger;

import android.app.Activity;
import android.content.ComponentName;
import android.content.Context;
import android.content.Intent;
import android.content.ServiceConnection;
import android.os.Bundle;
import android.os.Handler;
```

(continues on next page)

(continued from previous page)

```
import android.os.IBinder;
import android.os.Message;
import android.os.Messenger;
import android.os.RemoteException;
import android.os.SharedMemory;
import android.system.ErrnoException;
import android.widget.Toast;
import android.util.Log;

import org.jssec.android.shared.PkgCert;
import org.jssec.android.shared.SigPerm;
import org.jssec.android.shared.Utils;

import java.nio.ByteBuffer;
import java.nio.ReadOnlyBufferException;

import static android.system.OsConstants.PROT_EXEC;
import static android.system.OsConstants.PROT_READ;
import static android.system.OsConstants.PROT_WRITE;

public class MainActivity extends Activity {

    private final String TAG = "SHMClient";

    // Messenger used for sending data to Service
    private Messenger mServiceMessenger = null;

    // SharedMemory objects
    private SharedMemory myShared1;
    private SharedMemory myShared2;
    // ByteBuffers for mapping SharedMemories
    private ByteBuffer mBuf1;
    private ByteBuffer mBuf2;

    // Information of using Activity
    private static final String SHM_PACKAGE =
        "org.jssec.android.sharedmemory.inhouseservice.messenger";
    private static final String SHM_CLASS =
        "org.jssec.android.sharedmemory.inhouseservice.messenger.SHMService";

    // In-house Signature Permission
    private static final String MY_PERMISSION =
        "org.jssec.android.sharedmemory.inhouseservice.messenger.MY_PERMISSION";

    // Hash value of the certification of In-house applications
    private static String sMyCertHash = null;
    private static String myCertHash(Context context) {
        if (sMyCertHash == null) {
            if (Utils.isDebuggable(context)) {
                // Hash value of the certificate "androiddebugkey" stored in
                // debug.keystore
                sMyCertHash = "0EFB7236 328348A9 89718BAD DF57F544 D5CCB4AE␣
↪B9DB34BC 1E29DD26 F77C8255";
            } else {
                // Hash value of the certificate "my company key" in keystore
                sMyCertHash = "D397D343 A5CBC10F 4EDDEB7C A10062DE 5690984F␣
```

(continues on next page)

(continued from previous page)

```
↪1FB9E88B D7B3A7C2 42E142CA";
            }
        }
        return sMyCertHash;
    }

    // true iff connecting to Service
    private boolean mIsBound = false;

    // Handler handling Messages received from Server
    private class MyHandler extends Handler {
        @Override
        public void handleMessage(Message msg) {
            switch (msg.what) {
            case SHMService.SHMEM1:
                // SHMEM 1 is provided from Service
                // ShareMemory object is stored in Message.obj
                Log.d(TAG, "got SHMEM1");
                myShared1 = (SharedMemory) msg.obj;
                useSHMEM1();
                break;
            case SHMService.SHMEM2:
                // SHMEM2 is provided from Service
                Log.d(TAG, "got SHMEM2");
                myShared2 = (SharedMemory) msg.obj;
                useSHMEM2();
                break;
            case SHMService.MSG_END:
                Log.d(TAG, "got MSG_END");
                alloverNow();
                break;
            default:
                Log.e(TAG, "invalid message: " + msg.what);
            }
        }
    }

    private Handler mHandler = new MyHandler();

    // Messanger used when receiving data from Service
    private Messenger mLocalMessenger = new Messenger(mHandler);

    // Connection used for connecting to Service
    // This is needed if implementation uses bindService
    private class MyServiceConnection implements ServiceConnection {
        // called when connected with Service
        public void onServiceConnected(ComponentName className, IBinder service){
            mServiceMessenger = new Messenger(service);
            // When bound to SharedMemory Service, request 1st SharedMemory
            sendMessageToService(SHMService.MSG_ATTACH);
        }
        // This is called when Service unexpectedly terminate and connection is
        // broken
        public void onServiceDisconnected(ComponentName className){
            mIsBound = false;
            mServiceMessenger = null;
```

(continues on next page)

(continued from previous page)

```
        }
    }
    private MyServiceConnection mServiceConnection;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        doBindService ();
    }

    // Connect to Shared Memory Service
    private void doBindService () {
        mServiceConnection = new MyServiceConnection();
        if (!mIsBound) {
            // *** POINT 9 *** Verify that the in-house-defined signature
            // permission is defined by the in-house application.
            if (!SigPerm.test(this, MY_PERMISSION, myCertHash(this))) {
                Toast.makeText(this,  "In-house signature permission is not␣
↪defined by an in-house application.", Toast.LENGTH_LONG).show();
                return;
            }
            // *** POINT 10 *** Verify that the destination application is signed
            // by the in-house certificate.
            if (!PkgCert.test(this, SHM_PACKAGE, myCertHash(this))) {
                Toast.makeText(this, "Binding Service is not an in-house␣
↪application.", Toast.LENGTH_LONG).show();
                return;
            }
        }
        Intent it = new Intent();
        // *** POINT 11 *** Sensitive information can be sent because the
        // destination application is in-house.
        it.putExtra("PARAM", "Sensitive Information");

        // *** POINT 12 *** Use explicit Intent to call an in-house service
        it.setClassName(SHM_PACKAGE, SHM_CLASS);

        if (!bindService(it, mServiceConnection, Context.BIND_AUTO_CREATE)) {
            Toast.makeText(this, "Bind Service Failed", Toast.LENGTH_LONG).show();
            return;
        }
        mIsBound = true;
    }

    // Unbind connection with Service
    private void releaseService () {
        unbindService(mServiceConnection);
    }

    // An example of using SHMEM1
    private void useSHMEM1 () {
        // Because only read access is permitted for SHMEM1, mapping with
        // different protection mode will raise an exception.
        // The exception will be handled by mapMemory()
        mBuf1 = mapMemory(myShared1, PROT_WRITE, SHMService.SHMEM1_BUF1_OFFSET,
```

(continues on next page)

(continued from previous page)

```
                        SHMService.SHMEM1_BUF1_LENGTH);
        // map with PROT_READ
        mBuf1 = mapMemory(myShared1, PROT_READ, SHMService.SHMEM1_BUF1_OFFSET,
                        SHMService.SHMEM1_BUF1_LENGTH);
        // Read data which Service side set
        int len = mBuf1.getInt();
        byte[] bytes = new byte[len];
        mBuf1.get(bytes);
        String message = new String(bytes);
        Toast.makeText(MainActivity.this,
                    "Got: " + message, Toast.LENGTH_LONG).show();
        // Because the buffer is read only, writing will cause
        // ReadOnlyBufferException
        try {
            mBuf1.putInt(0);
        } catch (ReadOnlyBufferException e){
            Log.e(TAG, "Write to read only buffer: " + e.getMessage());
        }
        // Reply to Service
        sendMessageToService(SHMService.MSG_REPLY1);
        // then, request a SharedMemory with write permission
        sendMessageToService(SHMService.MSG_ATTACH2);
    }

    // An example of using SHMEM2
    private void useSHMEM2 () {
        // We are allowed to write into SHMEM2, map it with PROT_WRITE
        // Service side set SHMEM2 as PROT_WRITE, so mapping with
        // PROT_READ | PROT_WRITE will raise an exception
        mBuf2 = mapMemory(myShared2, PROT_WRITE, SHMService.SHMEM2_BUF1_OFFSET,
                        SHMService.SHMEM2_BUF1_LENGTH);
        if (mBuf2 != null) {
            // Even if the protection mode is PROT_WRITE only, it will also be
            // readable on most SoC.
            int size = mBuf2.getInt();
            byte [] bytes = new byte[size];
            mBuf2.get(bytes);
            String msg = new String(bytes);
            Log.d(TAG, "Got a message from service: " + msg);

            // Accessing outside of the mapped region will cause
            // IndexOutOfBoundsException
            try {
                mBuf2.get(SHMService.SHMEM2_BUF1_LENGTH + 1);
            } catch (IndexOutOfBoundsException e){
                Log.e(TAG, "out of bound: " + e.getMessage());
            }

            // Override the data which Service side set before
            String replyStr = "OK Thanks!";
            mBuf2.putInt(replyStr.length());
            mBuf2.put(replyStr.getBytes());
            // Reply to Service
            sendMessageToService(SHMService.MSG_REPLY2);
        }
    }
```

(continues on next page)

(continued from previous page)

```
    // Map specified SharedMemory
    private ByteBuffer mapMemory(SharedMemory mem, int proto, int offset,
                                 int length){
        ByteBuffer tempBuf;
        try {
            tempBuf = mem.map(proto, offset, length);
        } catch (ErrnoException e){
            Log.e(TAG,"could not map, proto: " + proto + ", offset:" +
                      offset +", length: " + length + "\n " + e.getMessage() +
                      "err no. = " + e.errno);
            return null;
        }
        return tempBuf;
    }

    // Reply message to Server
    private void sendMessageToService(int what){
        try {
            Message msg = Message.obtain();
            msg.what = what;
            msg.replyTo = mLocalMessenger;
            mServiceMessenger.send(msg);
        } catch (RemoteException e) {
            Log.e(TAG, "Error in sending message: " + e.getMessage());
        }
    }

    // Finalize no more used SharedMemory
    private void alloverNow() {
        // Notify Service that we are done
        sendMessageToService(SHMService.MSG_DETACH);
        sendMessageToService(SHMService.MSG_DETACH2);
        // unmap ByteBuffers
        if (mBuf1 != null) SharedMemory.unmap(mBuf1);
        if (mBuf2 != null) SharedMemory.unmap(mBuf2);
        // Close SharedMemory
        myShared1.close();
        myShared2.close();
        mBuf1 = null;
        mBuf2 = null;
        myShared1 = null;
        myShared2 = null;
        // Disconnect from Service
        releaseService();
    }
}
```

*** Point 13 *** When exporting an APK, sign the APK using the same developer key as the destination application.

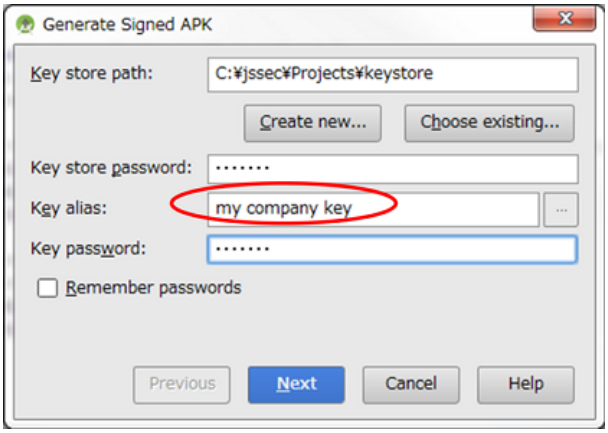


Fig. 4.11.5: Signing the APK with the same developer key as the destination application

4.11.3 Rule Book

When using SharedMemory, the rules contained in the rule book (4.4.2. *Rule Book*) for the service must be observed. In addition to the rule book, the following rules must also be observed.

1. *Permissions are set properly by the side providing the shared memory for allowing access by the using side (required)*
2. *All data in the shared memory is designed assuming that it will be read by sharing applications (required)*

4.11.3.1 Permissions are set properly by the side providing the shared memory for allowing access by the using side (required)

When memory is shared, in the design of operations allowable in the memory, each application must limit operations to the minimum required for preventing leaking, alteration, and corruption of information. Services that create SharedMemory objects can use SharedMemory#setProtect() to limit the allowable operations in the entire shared memory before sharing with other applications. The initial values for the operations allowable in the SharedMemory object are read, write, and execute. Except for special reasons, use of executable memory areas should be avoided in order to prevent execution of invalid code[38]. Also, if other applications need to write to the shared memory, a special-purpose shared memory is created and provided separately for enabling safe sharing of memory.

The argument of SharedMemory#setProtect() is a logical OR for the bit flags (PROT_READ, PROT_WRITE, PROT_EXEC) corresponding to read, write, and execute, respectively. An example is shown below for allowing reading and writing only for the SharedMemory object shMem.

```
shMem.setProtect(PROT_READ | PROT_WRITE)
```

SharedMemory#map() must be executed beforehand in order to enable access by the client to areas (all or part) within the shared memory. During this process, the allowable operations for the memory are specified by an argument, but operations cannot be specified above those permitted by the service beforehand using SharedMemory#setProtect(). For example, the client cannot specify write operations when the service permits reading only. An example is shown below where the SharedMemory object ashMem provided by the service performs map().

```
ByteBuffer mbuf;
// If the Service only allows READ from ashMem,
// the following code raises an exception
mbuf = ashMem.map(offset, length, PROT_WRITE);
```

At the client side, setProtect() can be called to redo the settings so that operations are allowed for the entire shared memory, but like map(), the settings cannot be made to allow operations above those that were permitted by the service.

[38] For some devices (based on the CPU architecture that is used), if a certain memory area is readable, it automatically becomes executable. However, even in these cases, writing can be prohibited for these areas to prevent writing of executable code in these areas by other applications.

4.11.3.2 All data in the shared memory is designed assuming that it will be read by sharing applications (required)

As described above, when memory is shared with other applications, the service can set the access permissions (read, write, execute) for the shared memory beforehand. However, even if the flag is set to PROT_WRITE only to allow writing only, in certain cases, reading of the memory cannot be prohibited. In other words, if the memory management unit (MMU) being used by the device does not support memory access that allows writing only, allowing writing for a certain memory area will also allow reading. It is thought that a large number of devices actually have this configuration, and as a result, design must be performed under the assumption that the contents of the shared memory will be known by other applications.

```
// Assume that Service side only allow writing go ashMem
// by SharedMemory#setProtect(PROT_WRITE).
// It is most of the case that even the client map with
// PROT_WRITE, he mapped buffer can be read.
ByteBuffer buf;
buf = ashMem.map(offset, length, PROT_WRITE);
// On most of SoC, read does not cause errors
int len = buf.getInt();
byte [] bytes = new byte[len];
buf.get(bytes);
```

Although PROT_NONE can be specified for the flag to prevent all operations, this defeats the purpose of having a shared memory.

4.11.4 Advanced Topics

4.11.4.1 Actual State of Shared Memory

Up to this point, the memory-sharing mechanism where memory was shared among multiple applications was described. However, in actuality, shared memory is a mechanism that shares the same physical memory area among multiple processes. Each process maps the shared physical memory area to its own address space for accessing (this is performed by SharedMemory#map()). For Android shared memory, the mapped memory area (for Java language) is a single ByteBuffer object. (If shared memory that exceeds the page size is allocated, typically, the shared physical memory area is not divided into consecutive areas, but instead, it is divided into multiple non-consecutive pages. However, if mapped onto a process address space, the memory area becomes consecutive address spaces.)

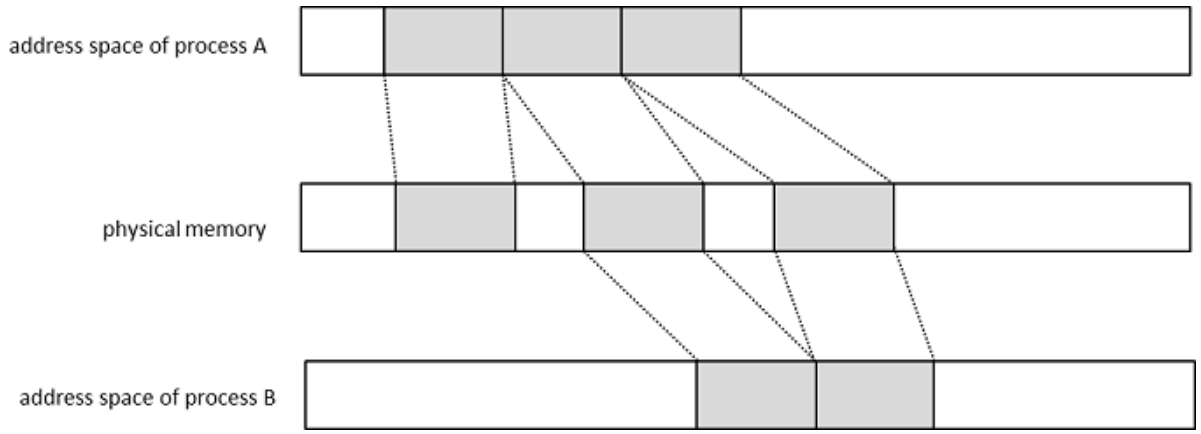


Fig. 4.11.6: Physical memory and process address space

In Unix-based OS, including the Android OS, the connected terminal, USB device, or other peripheral device is abstracted using the concept of device files, and the device is handled as a virtual file. Shared memory in the Android OS is not an exception to this, and this handling corresponds to the device file /dev/ashmem. When this device file

is opened, the file descriptor is returned in the same way as when a normal file is opened, and through this process, the shared memory is accessed. In the same way as normal files, this file descriptor can use mmap() to map to the process address space. In Unix-based OS, mmap() is the standard system call, and it obtains the file descriptors for devices files for a wide range of devices and provides a function for mapping the device to the address space of the calling process. This is also used for the shared memory of the Android OS. The mapped address space is visible as a byte sequence from the program (ByteBuffer for Java as mentioned above, and char * at the C language level).

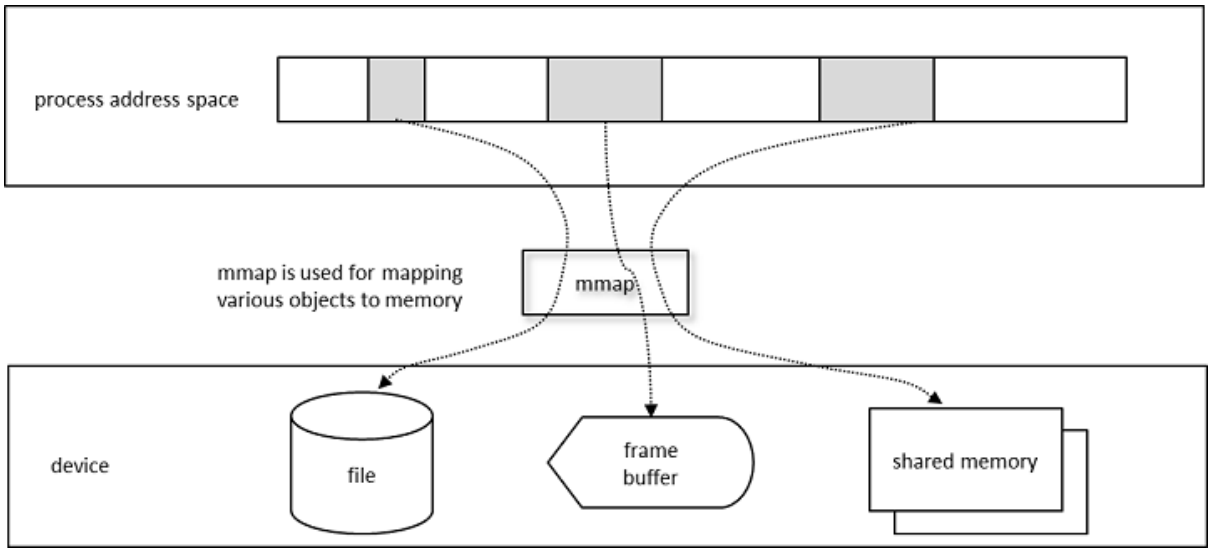


Fig. 4.11.7: Mapping of virtual file and address space

The sharing of memory between processes in this framework is equivalent to sharing the file descriptor of /dev/asmem corresponding to this memory area[39]. As a result, this enables low costs for sharing, and after mapping to the address space of the process, this enables access at the same efficiency as normal memory access.

4.11.4.2 Disabling sharedUserId in Newly Installed Apps

In Android, it is possible to share the permission status of declared permissions and file access permissions among multiple apps by using an element called sharedUserId, which is defined in the manifest file. This is because the same user ID is assigned to multiple apps in an Android device if the same values are defined for the parameters and the apps are signed with the same developer certificate.

This parameter was deprecated at API level 29, but no transition method was provided. If the parameter was deleted from the manifest file, an error would occur when updating the app, and once set, it could not be changed. As a result, despite the deprecation, sharedUserId was still set even in new installations.

On devices with Android 13 installed, newly installed apps are no longer affected by sharedUserId by defining the following element in the manifest file. However, this applies only to newly installed apps, and the user ID set in sharedUserId will continue to be used when updating existing apps.

```
android:sharedUserMaxSdkVersion
```

As the name suggests, this element specifies the maximum device version (API level) where sharedUserId is applied. This parameter is valid from Android 13, and so at the time of writing (July 2022), 32 should be specified if enabled for use.

[39] The file descriptor is a unique value within the process, and so when it is passed to other processes, proper conversion is required, but this does not need to be a consideration at the Android SDK API level.

4.12 Using NFC and FeliCa

NFC (Near Field Communication) and FeliCa are both short-range wireless technologies that communicate by holding a device over a reader or tag. Japan-specific FeliCa (Type-F) is included within the framework of NFC, which is an international standard. NFC roughly consists of the following types depending on communication methods and standards:

- Type-F (FeliCa): Mainly popular in Japan. Widely used for transportation IC cards such as Suica and PASMO, and electronic payments such as iD and QUICPay.
- Type-A: The most globally prevalent standard. Used in Japan for taspo, and overseas for public phones and public transit cards.
- Type-B: Boasts robust security and is used for My Number Cards, driver's licenses, passports, etc.
- NFC Contactless Payment: International standard contactless payment using Type-A/B. Used for credit card (Visa/Mastercard, etc.) contactless payments, Apple Pay, Google Wallet, etc.

NFC settings on Android devices can usually be toggled ON/OFF by the user from menus such as the following: Settings > Connected devices > Connection preferences > NFC

4.12.1 Sample Code

When developing an app that performs NFC/FeliCa communication, a safe and efficient implementation design is required that considers preventing processing delays caused by scanning unnecessary standards, preventing app crashes or accidental installation on non-equipped devices, and suppressing unnecessary background processing (preventing battery drain).

Here, using a simple reader app that detects FeliCa (Type-F) cards and retrieves and displays their basic information, "IDm (Manufacture ID)" and "PMm (Manufacture Parameter)", as an example, a safe approach for implementing NFC features is explained.

Points:

1. When starting reader mode (enableReaderMode), specify only the required standards (FLAG_READER_NFC_F when targeting FeliCa only) so as not to detect extra non-target standards.
2. If the use of NFC features is essential to the operation of the app, use Google Play Store's distribution filtering feature to exclude non-equipped devices in advance and prevent accidental installation.
3. Enable reader mode only when the app is in the foreground of the screen (onResume), and reliably disable it (disableReaderMode) when transitioning to the background (onPause).

AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools">

    <uses-permission android:name="android.permission.NFC" />
    <uses-feature android:name="android.hardware.nfc" android:required="true" />

    <application
        android:allowBackup="true"
        android:dataExtractionRules="@xml/data_extraction_rules"
        android:fullBackupContent="@xml/backup_rules"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/Theme.FeliCa_Sample">
        <activity
```

(continues on next page)

(continued from previous page)

```
            android:name=".MainActivity"
            android:exported="true"
            android:label="@string/app_name"
            android:theme="@style/Theme.FeliCa_Sample"
            android:windowSoftInputMode="adjustResize">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>

</manifest>
```

MainActivity.kt

```
/*
 * Copyright (C) 2012-2026 Japan Smartphone Security Association
 *
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 *
 *      http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */

package org.jssec.android.nfc.felica_sample

import android.nfc.NfcAdapter
import android.nfc.Tag
import android.nfc.tech.NfcF
import android.os.Bundle
import android.widget.Toast
import androidx.activity.ComponentActivity
import androidx.activity.compose.setContent
import androidx.activity.enableEdgeToEdge
import androidx.compose.foundation.layout.Arrangement
import androidx.compose.foundation.layout.Column
import androidx.compose.foundation.layout.Spacer
import androidx.compose.foundation.layout.fillMaxSize
import androidx.compose.foundation.layout.height
import androidx.compose.foundation.layout.padding
import androidx.compose.material3.MaterialTheme
import androidx.compose.material3.Scaffold
import androidx.compose.material3.Surface
import androidx.compose.material3.Text
import androidx.compose.runtime.Composable
import androidx.compose.runtime.getValue
import androidx.compose.runtime.mutableStateOf
import androidx.compose.runtime.setValue
```

(continues on next page)

(continued from previous page)

```
import androidx.compose.ui.Alignment
import androidx.compose.ui.Modifier
import androidx.compose.ui.text.font.FontWeight
import androidx.compose.ui.unit.dp

class MainActivity : ComponentActivity(), NfcAdapter.ReaderCallback {

    private var nfcAdapter: NfcAdapter? = null
    private var cardInfo by mutableStateOf<CardInfo?>(null)
    private var isScanning by mutableStateOf(false)

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        enableEdgeToEdge()

        // Obtain the device's default NfcAdapter
        nfcAdapter = NfcAdapter.getDefaultAdapter(this)
        if (nfcAdapter == null) {
            // Point 2 Error handling when APK is directly installed on a non-NFC␣
↪device via USB, etc.
            Toast.makeText(this, "NFC is not supported", Toast.LENGTH_LONG).show()
        }

        setContent {
            // Use general-purpose MaterialTheme to facilitate copy-paste reuse
            MaterialTheme {
                Scaffold(modifier = Modifier.fillMaxSize()) { innerPadding ->
                    FeliCaScannerScreen(
                        cardInfo = cardInfo,
                        isScanning = isScanning,
                        modifier = Modifier.padding(innerPadding)
                    )
                }
            }
        }
    }

    override fun onResume() {
        super.onResume()
        nfcAdapter?.let {
            isScanning = true
            val options = Bundle()

            // Point 1 Limit detection standards to FeliCa (FLAG_READER_NFC_F) and␣
↪configure to skip unnecessary processing such as NDEF checks
            // Point 3 Enable reader mode while the app is in the foreground␣
↪(onResume)
            it.enableReaderMode(
                this,
                this,
                NfcAdapter.FLAG_READER_NFC_F or NfcAdapter.FLAG_READER_SKIP_NDEF_␣
↪CHECK,
                options
            )
        }
    }
```

(continues on next page)

(continued from previous page)

```
    override fun onPause() {
        super.onPause()
        // Point 3 Reliably disable reader mode when the app transitions to the␣
↪background (onPause)
        nfcAdapter?.disableReaderMode(this)
        isScanning = false
    }

    // Callback invoked from a separate thread when a card is detected
    override fun onTagDiscovered(tag: Tag) {
        val nfcF = NfcF.get(tag)
        if (nfcF != null) {
            // Safely retrieve FeliCa-specific data (IDm, PMm)
            val idm = tag.id.toHexString()
            val pmm = nfcF.manufacturer.toHexString()
            cardInfo = CardInfo(idm = idm, pmm = pmm)
        }
    }

    // Extension function to convert byte array to hexadecimal string
    private fun ByteArray.toHexString(): String {
        return joinToString("") { "%02X".format(it) }
    }
}

// Data class holding detected card information
data class CardInfo(val idm: String, val pmm: String)

@Composable
fun FeliCaScannerScreen(
    cardInfo: CardInfo?,
    isScanning: Boolean,
    modifier: Modifier = Modifier
) {
    Surface(
        modifier = modifier.fillMaxSize(),
        color = MaterialTheme.colorScheme.background
    ) {
        Column(
            modifier = Modifier
                .fillMaxSize()
                .padding(16.dp),
            horizontalAlignment = Alignment.CenterHorizontally,
            verticalArrangement = Arrangement.Center
        ) {
            Text(
                text = "FeliCa Reader",
                style = MaterialTheme.typography.headlineMedium,
                fontWeight = FontWeight.Bold
            )
            Spacer(modifier = Modifier.height(32.dp))
            if (cardInfo != null) {
                CardDetails(cardInfo)
            } else {
                Text(
```

(continues on next page)

(continued from previous page)

```
                    text = if (isScanning) "Hold your card near the device" else
↪"NFC is disabled",
                    style = MaterialTheme.typography.bodyLarge
                )
            }
        }
    }
}

@Composable
fun CardDetails(cardInfo: CardInfo) {
    Column(
        horizontalAlignment = Alignment.CenterHorizontally
    ) {
        InfoRow(label = "IDm (Manufacture ID)", value = cardInfo.idm)
        Spacer(modifier = Modifier.height(16.dp))
        InfoRow(label = "PMm (Manufacture Parameter)", value = cardInfo.pmm)
    }
}

@Composable
fun InfoRow(label: String, value: String) {
    Column(horizontalAlignment = Alignment.CenterHorizontally) {
        Text(text = label, style = MaterialTheme.typography.labelMedium)
        Text(text = value, style = MaterialTheme.typography.bodyLarge, fontWeight␣
↪= FontWeight.Bold)
    }
}
```

4.12.2 Rule Book

When using NFC / FeliCa features, adhere to the following rules.

1. *In Apps Where NFC Features Are Essential to Operation, Restrict Distribution by Declaring required=”true” in the uses-feature Attribute (Required)*
2. *Enable and Disable Reader Mode (enableReaderMode) in Complete Synchronization with Activity Lifecycle (onResume/onPause) (Required)*
3. *When Starting enableReaderMode, Do Not Specify Unnecessary Non-Target Communication Standards (TypeA/B/F), and Set Only Flags Matching the Target (Required)*
4. *When Processing or Transmitting Retrieved Tag Data (IDm/PMm, Other Block Data, etc.) to an External Server, Verify Safety of Input Data (Required)*

4.12.2.1 In Apps Where NFC Features Are Essential to Operation, Restrict Distribution by Declaring required=”true” in the uses-feature Attribute (Required)

If the app has NFC or FeliCa features at its core and cannot operate without them, you must declare <uses-feature android:name=”android.hardware.nfc” android:required=”true” /> in AndroidManifest.xml to prevent installation on devices without NFC. Failure to do so allows users using devices without NFC to install the app from the store, causing NullPointerExceptions and app crashes when launching the app or using features. Also, in preparation for direct installations during development and testing (such as USB debugging), detect cases where NfcAdapter.getDefaultAdapter(this) returns null in code as well, and perform appropriate handling (such as displaying a warning message and suspending processing).

4.12.2.2 Enable and Disable Reader Mode (enableReaderMode) in Complete Synchronization with Activity Lifecycle (onResume/onPause) (Required)

Control of NFC devices via enableReaderMode must be limited only to periods when the app is located in the foreground directly visible to the user. Specifically, reader mode must be enabled in onResume() in the Activity lifecycle, and disabled by reliably calling disableReaderMode() when entering onPause(). If this lifecycle synchronization is incorrect, the NFC reading process will remain resident even while the app is in the background, causing abnormal battery consumption or obstructing normal operations of other apps with NFC features (such as payment processing).

4.12.2.3 When Starting enableReaderMode, Do Not Specify Unnecessary Non-Target Communication Standards (Type-A/B/F), and Set Only Flags Matching the Target (Required)

When calling enableReaderMode, you must explicitly specify flags matching the communication standard targeted for connection (FLAG_READER_NFC_F in the case of FeliCa) and opt out of scanning unnecessary standards. Also, for rapid reading and suppression of resource consumption, specifying FLAG_READER_SKIP_NDEF_CHECK together is strongly recommended in use cases where metadata integrity checks for NDEF (NFC Data Exchange Format) are unnecessary. If all standards are targeted for scanning without restricting target standards, hypersensitive sensor reactions or unnecessary polling occur, increasing risks of reacting to tags of unintended standards or degrading user experience due to increased overhead of the scan process itself.

4.12.2.4 When Processing or Transmitting Retrieved Tag Data (IDm/PMm, Other Block Data, etc.) to an External Server, Verify Safety of Input Data (Required)

IDm, PMm read from cards via NFC connection, or various data read from secure/non-secure areas must all be regarded as "untrusted input data (Input)" passed from the outside. When parsing and processing these data inside the app, passing them to integration processing with Web servers, or specifying them as query conditions for a database (SQLite), always perform "Input Validation" based on general secure coding principles to prevent SQL injection, buffer overflows, inclusion of control characters, etc.

4.12.3 Advanced

4.12.3.1 Considerations for Secure Data Reading and Countermeasures Against Tampering in FeliCa Cards

The method of performing user identification by simply retrieving "IDm (Manufacture ID)", which is a fixed identifier, from a FeliCa card (or a Type-B card such as a My Number Card) is adopted in many time and attendance management systems, reception systems, etc., because it is extremely simple. However, it is necessary to properly understand the fundamental security characteristic that "IDm is public data like a serial number printed on the card surface, and spoofing (creation of forged cards) is easily possible."

If implementing processes that require a certain level of security or higher in an app, such as identity verification, cashless payment, or access restriction to important areas, avoid designs that rely solely on IDm, which is public information, and adopt one of the following approaches.

Mutual Authentication Using Encryption Keys and Implementation of Encrypted Communication

When accessing the "secure area (area protected by system codes or service codes)" of FeliCa, design the system to execute a secure mutual authentication process using encryption keys between the card and reader (app/server), and perform reading and writing of data using a temporary key generated for each session.

Verification of Authenticity (Integrity) via Digital Signatures

When reading the basic 4 personal information fields from a My Number Card (Type-B) or similar cases, implement logic on the app/server side to verify the digital signature of the "Public Key Infrastructure for Individual Number Cards" attached to the data itself recorded on the card or a signature (integrity) from a trusted issuer, reliably eliminating spoofing and tampering.

5 How to use Security Functions

Android provides various security features, such as encryption, digital signatures, and permissions. If these security features are not handled correctly, they will not function effectively, creating potential vulnerabilities. This chapter presents articles assuming scenarios where developers utilize these security features.

5.1 Creating Authentication (Password/Passkey) Screens

In traditional Android app development, it was common to implement the UI of the password input screen uniquely and manage the entered information on one's own. However, today, to balance security and convenience, a modern authentication approach is recommended that delegates to the OS standard authentication framework, such as using Credential Manager to securely handle not only passwords but also passkeys and Google sign-in with a single API.

This section explains the basic display rules to follow when implementing a legacy password input screen, display protection for sensitive information (disabling screen capture), and secure design policies when handling credentials.

5.1.1 Sample Code

The following describes points to consider regarding security when creating screens that handle credentials such as passwords.

Points:

1. Provide a mask display (displaying as ●) feature when entering a password
2. Disable screen capture using FLAG_SECURE on sensitive screens such as password screens
3. Do not hardcode passwords or cryptographic secrets within the app
4. Use standard APIs such as Credential Manager to handle credentials securely

In the following layout XML, the mask display is configured for the password input field.

activity_password.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    android:padding="32dp"
    android:gravity="center">

    <TextView
```

(continues on next page)

(continued from previous page)

```
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Login Authentication"
        android:textSize="24sp"
        android:textStyle="bold"
        android:layout_marginBottom="32dp" />

    <!-- Point 1 Providing a mask display (displaying as ) feature by setting␣
↪android:inputType="textPassword" -->
    <EditText
        android:id="@+id/passwordEditText"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:hint="Password (or PIN)"
        android:inputType="textPassword"
        android:layout_marginBottom="24dp" />

    <Button
        android:id="@+id/loginButton"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Authenticate" />

</LinearLayout>
```

The following Activity shows the screen protection settings and the basic concept when calling a modern authentication API.

PasswordActivity.kt

```
/*
 * Copyright (C) 2012-2026 Japan Smartphone Security Association
 *
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 *
 *      http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */

package org.jssec.android.password.passwordinputui

import android.os.Bundle
import android.view.WindowManager
import android.widget.Button
import android.widget.EditText
import androidx.appcompat.app.AppCompatActivity

class PasswordActivity : AppCompatActivity() {

    override fun onCreate(savedInstanceState: Bundle?) {
```

(continues on next page)

(continued from previous page)

```
        super.onCreate(savedInstanceState)

        // Point 2 Disable screen capture using FLAG_SECURE on sensitive screens␣
↪such as password screens
        // *This is a mandatory security requirement strongly required by Google␣
↪Play policy.
        window.addFlags(WindowManager.LayoutParams.FLAG_SECURE)

        // Load the layout file activity_password.xml
        setContentView(R.layout.activity_password)

        // Point 1 Providing a mask display (displaying as ) feature by setting␣
↪android:inputType="textPassword" on the XML side
        val passwordEditText = findViewById<EditText>(R.id.passwordEditText)
        val loginButton = findViewById<Button>(R.id.loginButton)

        loginButton.setOnClickListener {
            // Point 3 Do not hardcode passwords or cryptographic secrets within␣
↪the app
            // Credentials should be obtained from user input or secure KeyStore,␣
↪and not embedded as fixed strings in the source code.

            // Point 4 Use standard APIs such as Credential Manager to handle␣
↪credentials securely
            // Instead of implementing unique, complex authentication logic on␣
↪your own, use OS standard APIs such as Credential Manager
            // to execute the save/retrieve flow for passkeys and passwords.
            performAuthentication()
        }
    }

    private fun performAuthentication() {
        // Implement the process to call Credential Manager (such as␣
↪GetCredentialRequest) here
    }
}
```

5.1.2 Rule Book

Be sure to follow the rules below when creating an authentication screen and handling credentials.

1. *Provide a mask display (displaying as) feature when entering a password (Required)*
2. *Disable screen capture using FLAG_SECURE on sensitive screens such as password screens (Required)*
3. *Do not hardcode passwords or cryptographic secrets within the app (Required)*
4. *Use standard APIs such as Credential Manager to handle credentials securely (Recommended)*

5.1.2.1 Provide a mask display (displaying as) feature when entering a password (Required)

Smartphones are often used in crowded places, and there is a high risk of passwords being shoulder-surfed by third parties. As an app specification, a feature to mask the password display is required. When specifying in layout XML, specify “textPassword” for the android:inputType attribute. When switching dynamically in program code, add InputType.TYPE_TEXT_VARIATION_PASSWORD using the setInputType() method of the EditText class.

5.1.2.2 Disable screen capture using FLAG_SECURE on sensitive screens such as password screens (Required)

On password input screens, sensitive information such as passwords is displayed on the screen. On such screens, there is a risk of personal information leaking due to screen recording by malware or screen capture by a third party, so it is necessary to disable screen capture.

Disabling screen capture can be achieved by setting addFlags(WindowManager.LayoutParams.FLAG_SECURE) on WindowManager. The Google Play developer program policy (Device and Network Abuse) also explicitly prohibits creating or facilitating means to bypass the FLAG_SECURE setting of other apps for security and privacy protection. The use of FLAG_SECURE on sensitive screens is a mandatory security requirement to protect apps and users.

5.1.2.3 Do not hardcode passwords or cryptographic secrets within the app (Required)

Credentials such as passwords, API keys, and encryption keys must not be hardcoded in the app's source code or asset files. Hardcoded secrets can be easily obtained by attackers using reverse engineering tools, causing major security risks. If system-level credentials are required, it is mandatory to use the KeyChain API, and credentials unique to each app must be saved using the Android KeyStore provider.

5.1.2.4 Use standard APIs such as Credential Manager to handle credentials securely (Recommended)

Rather than saving passwords on your own or intricately building complex dummy display logic, you should actively utilize the OS standard authentication framework such as Credential Manager, which can cooperate with passkeys and biometric authentication. By delegating the processing to standard APIs, user convenience can be improved while maintaining a secure design.

5.1.3 Advanced

5.1.3.1 Login process

The typical example where password input is required is the login process. The following describes several points to be careful of in the login process.

Error message when login fails

In the login process, two pieces of information, ID (account) and password, are entered. There are two cases when login fails: when the ID does not exist, and when the ID exists but the password is incorrect. If the login failure message distinguishes and displays these two cases, attackers can guess "whether the specified ID exists or not." To prevent such guessing, the message on login failure should display as follows without distinguishing the two cases.

Message example: The login ID or password is incorrect.

Auto-login feature

There is a feature that, once the login process succeeds, omits entering the login ID and password from the next time onwards to log in automatically. The auto-login feature improves convenience because troublesome input can be omitted, but on the other hand, it entails a risk of being abused by a third party if the smartphone is stolen.

The auto-login feature can be used only when the damage in case of abuse by a third party is acceptable, or when sufficient security measures are possible. For example, in the case of online banking apps, if a third party operates the device, financial damage will occur, so security measures are mandatory in conjunction with the auto-login feature. Measures include "requiring password re-entry immediately before financial transactions such as payment occur," and "thoroughly alerting the user when setting up auto-login to prompt secure device locking." When using auto-login, a careful study should be conducted considering convenience and risks based on these measures.

5.1.3.2 Changing passwords

When changing a password that has been set to a different password, the following input items should be prepared on the screen.

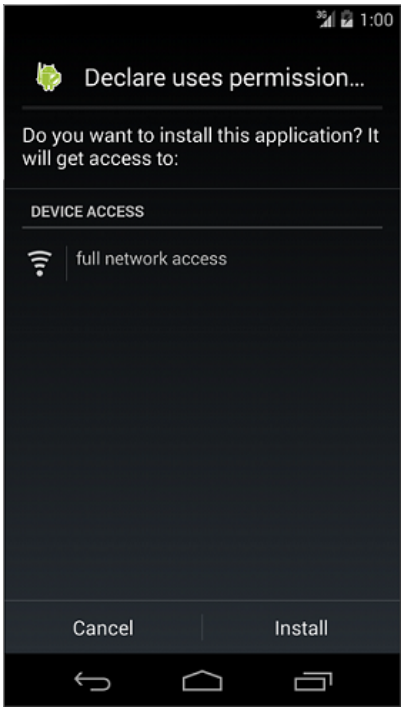
- Current password
- New password
- New password (for confirmation)

If the auto-login feature is included, there is a possibility that a third party can use the app. In that case, to prevent the password from being changed without permission, it is necessary to require the input of the current password. Also, to reduce the risk of becoming unable to use due to input errors of the new password, the new password should be requested twice.

5.1.3.3 The system "Show passwords" setting menu

In the Android settings menu, there is a setting called "Show passwords." Settings > Security > Show passwords

Up to versions prior to Android 5.0, this can be configured by this procedure. However, in Android 5.0 and later, "Show passwords" has been changed from a checkbox to a toggle button.



Settings - Show passwords

When the "Show passwords" setting is turned on, the last character entered is displayed as plain text. After a certain period of time (about 2 seconds) has passed, or when the next character is entered, the character displayed as plain text is masked. When turned off, it is masked immediately after entry. This is a setting that affects the entire system and is applied to all apps that use the password display feature of EditText.

Display of characters when entering passwords

5.1.3.4 Screen Recording Detection

From Android 15, a feature to detect screen recording in apps has been added.

While screen recording is running, a callback is invoked when the app transitions to visible or invisible. If an activity owned by the UID of the registered process is being recorded, the app is considered visible. It has become possible to notify users when screen recording is detected during sensitive operations on the app.

For example, when the screen is being recorded by user operation or another app, before displaying screens such as entering an ID and password or displaying personal information on the app, you can display a warning to the user or prohibit app operation to prevent unintended recording or distribution of sensitive or personal information.

An implementation example is as follows.

```
<uses-permission android:name="android.permission.DETECT_SCREEN_RECORDING" />
```

MainActivity.kt

```
private val mCallback = Consumer<Int> { state ->
    if (state == SCREEN_RECORDING_STATE_VISIBLE) {
        Log.i("TAG", "We're being recorded")
    } else {
        Log.i("TAG", "We're not being recorded")
    }
}

override fun onStart() {
    super.onStart()
    val initialState =
        windowManager.addScreenRecordingCallback(mainExecutor, mCallback)
    mCallback.accept(initialState)
}

override fun onStop() {
    super.onStop()
    windowManager.removeScreenRecordingCallback(mCallback)
}
```

5.1.3.5 Integrate Credential Manager with Autofill

By utilizing Android 15 (and the androidx.credentials library), developers can directly link specific Views (e.g., username and password input fields) to Credential Manager requests. When the user focuses on these Views, the corresponding requests are sent to the Credential Manager, and the generated credentials are aggregated across providers and displayed directly in the autofill UI (such as keyboard inline suggestions or pull-down suggestions). This feature can also be used as a fallback when the user accidentally closes the account selection tool of the Credential Manager.

The Jetpack androidx.credentials library is recommended as a recommended endpoint for the following reasons:

- **Integration**: androidx.credentials is part of Android Jetpack and integrates seamlessly with other Jetpack libraries.
- **Maintainability**: Officially supported by Google, with regular updates and bug fixes, making it highly reliable.
- **Security**: The latest security best practices for managing credentials are implemented to ensure secure credential management.
- **Simplicity**: Designed for easy implementation by developers, reducing code complexity.

Implementation Procedure

1. Create a GetCredentialRequest

```
val getPasswordOption = GetPasswordOption()
val getPublicKeyCredentialOption = GetPublicKeyCredentialOption(
    requestJson = requestJson
)
val getCredRequest = GetCredentialRequest(
    listOf(getPasswordOption, getPublicKeyCredentialOption)
)
```

2. Call the getCredential API

```
coroutineScope.launch {
    try {
        val result = credentialManager.getCredential(
            context = activityContext, // Use an activity-based context
            request = getCredRequest
        )
        handleSignIn(result)
    } catch (GetCredentialException e) {
        handleFailure(e)
    }
}
```

3. Set up autofill functionality in the view

```
import androidx.credentials.PendingGetCredentialRequest

usernameEditText.pendingGetCredentialRequest = PendingGetCredentialRequest(
    getCredRequest) { response -> handleSignIn(response)
}
passwordEditText.pendingGetCredentialRequest = PendingGetCredentialRequest(
    getCredRequest) { response -> handleSignIn(response)
}
```

Precautions

- setPendingGetCredentialRequest is an extension API of the androidx.credentials library, and the calling method differs between Kotlin and Java.
- Currently, this feature is available only on View objects.

5.1.3.6 Hide Passwords from Physical Devices

In Android 17, as part of security enhancements, the display behavior when entering passwords from physical devices (such as USB or Bluetooth-connected external keyboards) has been changed.

In traditional Android (Android 16 or earlier), when characters were entered in the password input field, a mechanism was common where “only the last character entered was displayed for a moment and then immediately changed to ● (mask)” in accordance with the system “Show passwords” setting for the user’s input confirmation. This behavior applied similarly not only to the on-screen software keyboard but also to input from physical keyboards in some cases, raising the risk of being peered from the surroundings (shoulder surfing) on tablet devices with large displays.

In contrast, in Android 17, the behavior has been revised to reduce risk, and when entering from a physical keyboard, it has been changed to not display password characters even temporarily (fully masking from the beginning).

This display behavior is managed by ShowSecretsSetting introduced in Android 17, and the display is controlled independently for “touch input (soft keyboard)” and “physical input (hardware keyboard).” It is important for developers to use standard input components that respect the OS security policies and configuration APIs (such as shouldShowPhysicalInput()) rather than implementing unique password display mechanisms on the app side.

5.2 Permission and Protection Level

There are four types of Protection Level within permission and they consist of normal, dangerous, signature, and signatureOrSystem. In addition, "development", "system", and "appop" exist, but since they are not used in general applications, explanation in this chapter is omitted. Depending on the Protection Level, permission is referred to as normal permission, dangerous permission, signature permission, or signatureOrSystem permission. In the following sections, such names are used.

5.2.1 Sample Code

5.2.1.1 How to Use System Permissions of Android OS

Android OS has a security mechanism called "permission" that protects its user's assets such as contacts and a GPS feature from a malware. When an application seeks access to such information and/or features, which are protected under Android OS, the application needs to explicitly declare a permission in order to access them. When an application, which has declared a permission that needs user's consent to be used, is installed, the following confirmation screen appears[1].

[1] In Android 6.0 (API Level 23) and later, the granting or refusal of user permissions does not occur when an app is installed, but instead at runtime when then app requests permissions. For more details, see Section "5.2.1.4. *Methods for using Dangerous Permissions in Android 6.0 and later*" and Section "5.2.3.6. *Modifications to the Permission model specifications in Android versions 6.0 and later*".

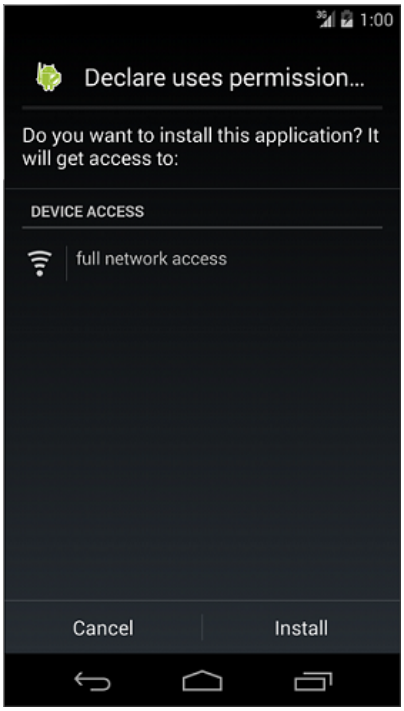


Fig. 5.2.1: Declare uses permission

From this confirmation screen, a user is able to know which types of features and/or information an application is trying to access. If the behavior of an application is trying to access features and/or information that are clearly unnecessary, then there is a high possibility that the application is a malware. Hence, as your application is not suspected to be a malware, declarations of permission to use needs to be minimized.

Points:

1. Declare a permission used in an application with uses-permission.
2. Do not declare any unnecessary permissions with uses-permission.

AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
          >

  <!-- *** POINT 1 *** Declare a permission used in an application with uses-
↪permission -->
  <!-- Permission to access Internet -->
  <uses-permission android:name="android.permission.INTERNET"/>

  <!-- *** POINT 2 *** Do not declare any unnecessary permissions with uses-
↪permission -->
  <!-- If declaring to use Permission that is unnecessary for application␣
↪behaviors, it gives users a sense of distrust. -->

  <application
      android:allowBackup="false"
      android:icon="@drawable/ic_launcher"
      android:label="@string/app_name" >
    <activity
        android:name=".MainActivity"
        android:label="@string/app_name"
```

(continues on next page)

(continued from previous page)

```
        android:exported="true" >
      <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER" />
      </intent-filter>
    </activity>
  </application>

</manifest>
```

5.2.1.2 How to Communicate Between In-house Applications with In-house-defined Signature Permission

Besides system permissions defined by Android OS, an application can define its own permissions as well. If using an in-house-defined permission (it is an in-house-defined signature permission to be more precise), you can build a mechanism where only communications between in-house applications is permitted. By providing the composite function based on inter-application communication between multiple in-house applications, the applications get more attractive and your business could get more profitable by selling them as series. It is a case of using in-house-defined signature permission.

The sample application "In-house-defined Signature Permission (UserApp)" launches the sample application "In-house-defined Signature Permission (ProtectedApp)" with Context.startActivity() method. Both applications need to be signed with the same developer key. If keys for signing them are different, the UserApp sends no Intent to the ProtectedApp, and the ProtectedApp processes no Intent received from the UserApp. Furthermore, it prevents malwares from circumventing your own signature permission using the matter related to the installation order as explained in the Advanced Topic section.

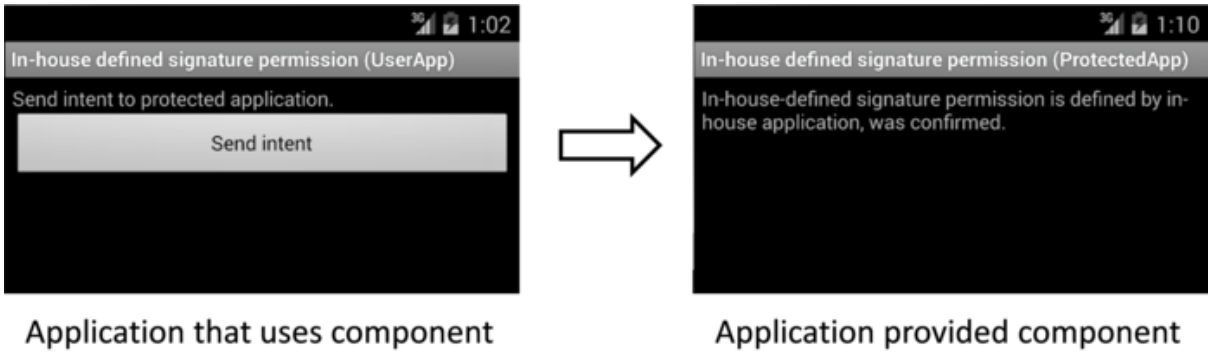


Fig. 5.2.2: Communication Between In-house Applications with In-house-defined Signature Permission

Points: Application Providing Component

1. Define a permission with protectionLevel="signature".
2. For a component, enforce the permission with its permission attribute.
3. If the component is an activity, you must define no intent-filter.
4. At run time, verify if the signature permission is defined by itself on the program code.
5. When exporting an APK, sign the APK with the same developer key that applications using the component use.

```
AndroidManifest.xml
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
```

(continues on next page)

(continued from previous page)

```
          >

  <!-- *** POINT 1 *** Define a permission with protectionLevel="signature" -->
  <permission
      android:name="org.jssec.android.permission.protectedapp.MY_PERMISSION"
      android:protectionLevel="signature" />

  <application
      android:allowBackup="false"
      android:icon="@drawable/ic_launcher"
      android:label="@string/app_name" >

    <!-- *** POINT 2 *** For a component, enforce the permission with its␣
↪permission attribute -->
    <activity
        android:name=".ProtectedActivity"
        android:exported="true"
        android:label="@string/app_name"
        android:permission="org.jssec.android.permission.protectedapp.MY_PERMISSION
↪" >

      <!-- *** POINT 3 *** If the component is an activity, you must define no␣
↪intent-filter -->
    </activity>
  </application>
</manifest>
```

ProtectedActivity.java

```
/*
 * Copyright (C) 2012-2026 Japan Smartphone Security Association
 *
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 *
 *      http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */

package org.jssec.android.permission.protectedapp;

import org.jssec.android.shared.SigPerm;
import org.jssec.android.shared.Utils;

import android.app.Activity;
import android.content.Context;
import android.os.Bundle;
import android.widget.TextView;

public class ProtectedActivity extends Activity {
```

(continues on next page)

(continued from previous page)

```
    // In-house Signature Permission
    private static final String MY_PERMISSION =
            "org.jssec.android.permission.protectedapp.MY_PERMISSION";

    // Hash value of in-house certificate
    private static String sMyCertHash = null;
    private static String myCertHash(Context context) {
        if (sMyCertHash == null) {
            if (Utils.isDebuggable(context)) {
                // Certificate hash value of "androiddebugkey" of debug.keystore
                sMyCertHash = "0EFB7236 328348A9 89718BAD DF57F544 D5CCB4AE␣
↪B9DB34BC 1E29DD26 F77C8255";
            } else {
                // Certificate hash value of "my company key" of keystore
                sMyCertHash = "D397D343 A5CBC10F 4EDDEB7C A10062DE 5690984F␣
↪1FB9E88B D7B3A7C2 42E142CA";
            }
        }
        return sMyCertHash;
    }

    private TextView mMessageView;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        mMessageView = (TextView) findViewById(R.id.messageView);

        // *** POINT 4 *** At run time, verify if the signature permission is
        // defined by itself on the program code
        if (!SigPerm.test(this, MY_PERMISSION, myCertHash(this))) {
            mMessageView.setText("In-house defined signature permission is not␣
↪defined by in-house application.");
            return;
        }

        // *** POINT 4 *** Continue processing only when the certificate matches
        mMessageView.setText("In-house-defined signature permission is defined by␣
↪in-house application, was confirmed.");
    }
}
```

SigPerm.java

```
/*
 * Copyright (C) 2012-2026 Japan Smartphone Security Association
 *
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 *
 *      http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
```

(continues on next page)

(continued from previous page)

```
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */

package org.jssec.android.shared;

import android.content.Context;
import android.content.pm.PackageManager;
import android.content.pm.PackageManager.NameNotFoundException;
import android.content.pm.PermissionInfo;
import android.os.Build;

import static android.content.pm.PackageManager.CERT_INPUT_SHA256;

public class SigPerm {

    public static boolean test(Context ctx, String sigPermName,
                               String correctHash) {
        if (correctHash == null) return false;
        correctHash = correctHash.replaceAll(" ", "");
        try {
            // Get the package name of the application which declares a permission
            // named sigPermName.
            PackageManager pm = ctx.getPackageManager();
            PermissionInfo pi =
                pm.getPermissionInfo(sigPermName, PackageManager.GET_META_DATA);
            String pkgname = pi.packageName;
            // Fail if the permission named sigPermName is not a Signature
            // Permission
            if (pi.protectionLevel != PermissionInfo.PROTECTION_SIGNATURE)
                return false;

            // Compare the actual hash value of pkgname with the correct hash
            // value.
            if (Build.VERSION.SDK_INT >= 28) {
                // ** if API Level >= 28, direct check is possible
                return pm.hasSigningCertificate(pkgname,
                                                Utils.hex2Bytes(correctHash),
                                                CERT_INPUT_SHA256);
            } else {
                // else(API Level < 28) use the facility of PkgCert
                return correctHash.equals(PkgCert.hash(ctx, pkgname));
            }

        } catch (NameNotFoundException e) {
            return false;
        }
    }
}
```

PkgCertWhitelists.java

```
/*
 * Copyright (C) 2012-2026 Japan Smartphone Security Association
 *
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
```

(continues on next page)

(continued from previous page)

```
 * You may obtain a copy of the License at
 *
 *      http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */

package org.jssec.android.shared;

import android.content.pm.PackageManager;
import java.util.HashMap;
import java.util.Map;
import android.content.Context;
import android.os.Build;

import static android.content.pm.PackageManager.CERT_INPUT_SHA256;

public class PkgCertWhitelists {
    private Map<String, String> mWhitelists = new HashMap<String, String>();

    public boolean add(String pkgname, String sha256) {
        if (pkgname == null) return false;
        if (sha256 == null) return false;

        sha256 = sha256.replaceAll(" ", "");
        if (sha256.length() != 64)
            return false;  // SHA-256 -> 32 bytes -> 64 chars
        sha256 = sha256.toUpperCase();
        if (sha256.replaceAll("[0-9A-F]+", "").length() != 0)
            return false;  // found non hex char

        mWhitelists.put(pkgname, sha256);
        return true;
    }

    public boolean test(Context ctx, String pkgname) {
        // Get the correct hash value which corresponds to pkgname.
        String correctHash = mWhitelists.get(pkgname);

        // Compare the actual hash value of pkgname with the correct hash value.
        if (Build.VERSION.SDK_INT >= 28) {
            // ** if API Level >= 28, direct checking is possible
            PackageManager pm = ctx.getPackageManager();
            return pm.hasSigningCertificate(pkgname,
                                            Utils.hex2Bytes(correctHash),
                                            CERT_INPUT_SHA256);
        } else {
            // else use the facility of PkgCert
            return PkgCert.test(ctx, pkgname, correctHash);
        }
    }
}
```

*** Point 5 *** When exporting an APK, sign the APK with the same developer key that applications using the component have used.

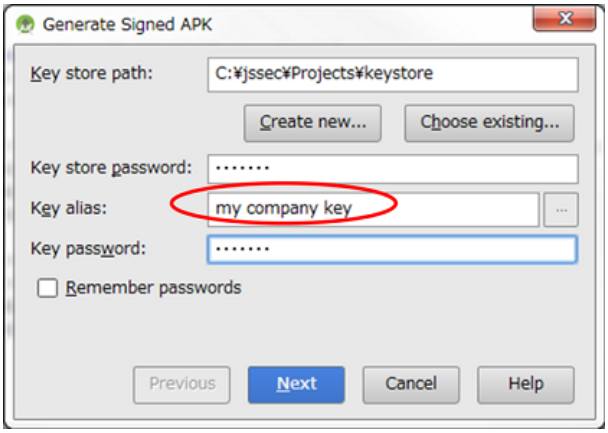


Fig. 5.2.3: Sign the APK with the same developer key that applications using the component have used

Points: Application Using Component

6. The same signature permission that the application uses must not be defined.
7. Declare the in-house permission with uses-permission tag.
8. Verify if the in-house signature permission is defined by the application that provides the component on the program code.
9. Verify if the destination application is an in-house application.
10. Use an explicit intent when the destination component is an activity.
11. When exporting an APK by [Build] -> [Generate Signed APK], sign the APK with the same developer key that the destination application uses.

```
AndroidManifest.xml
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
          >

  <queries>
    <package android:name="org.jssec.android.permission.protectedapp" />
  </queries>

  <!-- *** POINT 6 *** The same signature permission that the application uses␣
↪must not be defined -->

  <!-- *** POINT 7 *** Declare the in-house permission with uses-permission tag -->
  <uses-permission
      android:name="org.jssec.android.permission.protectedapp.MY_PERMISSION" />

  <application
      android:allowBackup="false"
      android:icon="@drawable/ic_launcher"
      android:label="@string/app_name" >
    <activity
        android:name=".UserActivity"
        android:label="@string/app_name"
        android:exported="true" >
      <intent-filter>
```

(continues on next page)

(continued from previous page)

```
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER" />
      </intent-filter>
    </activity>
  </application>
</manifest>
```

UserActivity.java

```
/*
 * Copyright (C) 2012-2026 Japan Smartphone Security Association
 *
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 *
 *      http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */

package org.jssec.android.permission.userapp;

import org.jssec.android.shared.PkgCert;
import org.jssec.android.shared.SigPerm;
import org.jssec.android.shared.Utils;

import android.app.Activity;
import android.content.Context;
import android.content.Intent;
import android.os.Bundle;
import android.view.View;
import android.widget.Toast;

public class UserActivity extends Activity {

    // Requested (Destination) application's Activity information
    private static final String TARGET_PACKAGE =
            "org.jssec.android.permission.protectedapp";
    private static final String TARGET_ACTIVITY =
            "org.jssec.android.permission.protectedapp.ProtectedActivity";

    // In-house Signature Permission
    private static final String MY_PERMISSION =
            "org.jssec.android.permission.protectedapp.MY_PERMISSION";

    // Hash value of in-house certificate
    private static String sMyCertHash = null;
    private static String myCertHash(Context context) {
        if (sMyCertHash == null) {
            if (Utils.isDebuggable(context)) {
                // Certificate hash value of "androiddebugkey" of debug.keystore.
                sMyCertHash = "0EFB7236 328348A9 89718BAD DF57F544 D5CCB4AE␣
```

(continues on next page)

(continued from previous page)

```
↪B9DB34BC 1E29DD26 F77C8255";
            } else {
                // Certificate hash value of "my company key" of keystore.
                sMyCertHash = "D397D343 A5CBC10F 4EDDEB7C A10062DE 5690984F␣
↪1FB9E88B D7B3A7C2 42E142CA";
            }
        }
        return sMyCertHash;
    }

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
    }

    public void onSendButtonClicked(View view) {

        // *** POINT 8 *** Verify if the in-house signature permission is defined
        // by the application that provides the component on the program code.

        if (!SigPerm.test(this, MY_PERMISSION, myCertHash(this))) {
            Toast.makeText(this, "In-house-defined signature permission is not␣
↪defined by In house application.", Toast.LENGTH_LONG).show();
            return;
        }

        // *** POINT 9 *** Verify if the destination application is an in-house
        // application.
        if (!PkgCert.test(this, TARGET_PACKAGE, myCertHash(this))) {
            Toast.makeText(this, "Requested (Destination) application is not in-
↪house application.", Toast.LENGTH_LONG).show();
            return;
        }

        // *** POINT 10 *** Use an explicit intent when the destination component
        // is an activity.
        try {
            Intent intent = new Intent();
            intent.setClassName(TARGET_PACKAGE, TARGET_ACTIVITY);
            startActivity(intent);
        } catch(Exception e) {
            Toast.makeText(this,
                           String.format("Exception occurs:%s", e.getMessage()),
                           Toast.LENGTH_LONG).show();
        }
    }
}
```

PkgCertWhitelists.java

```
/*
 * Copyright (C) 2012-2026 Japan Smartphone Security Association
 *
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
```

(continues on next page)

(continued from previous page)

```
 *
 *      http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */

package org.jssec.android.shared;

import android.content.pm.PackageManager;
import java.util.HashMap;
import java.util.Map;
import android.content.Context;
import android.os.Build;

import static android.content.pm.PackageManager.CERT_INPUT_SHA256;

public class PkgCertWhitelists {
    private Map<String, String> mWhitelists = new HashMap<String, String>();

    public boolean add(String pkgname, String sha256) {
        if (pkgname == null) return false;
        if (sha256 == null) return false;

        sha256 = sha256.replaceAll(" ", "");
        if (sha256.length() != 64)
            return false;  // SHA-256 -> 32 bytes -> 64 chars
        sha256 = sha256.toUpperCase();
        if (sha256.replaceAll("[0-9A-F]+", "").length() != 0)
            return false;  // found non hex char

        mWhitelists.put(pkgname, sha256);
        return true;
    }

    public boolean test(Context ctx, String pkgname) {
        // Get the correct hash value which corresponds to pkgname.
        String correctHash = mWhitelists.get(pkgname);

        // Compare the actual hash value of pkgname with the correct hash value.
        if (Build.VERSION.SDK_INT >= 28) {
            // ** if API Level >= 28, direct checking is possible
            PackageManager pm = ctx.getPackageManager();
            return pm.hasSigningCertificate(pkgname,
                                            Utils.hex2Bytes(correctHash),
                                            CERT_INPUT_SHA256);
        } else {
            // else use the facility of PkgCert
            return PkgCert.test(ctx, pkgname, correctHash);
        }
    }
}
```

PkgCert.java

```java
/*
 * Copyright (C) 2012-2026 Japan Smartphone Security Association
 *
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 *
 *      http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */

package org.jssec.android.shared;

import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;

import android.content.Context;
import android.content.pm.PackageInfo;
import android.content.pm.PackageManager;
import android.content.pm.PackageManager.NameNotFoundException;
import android.content.pm.Signature;

public class PkgCert {

    public static boolean test(Context ctx, String pkgname, String correctHash) {
        if (correctHash == null) return false;
        correctHash = correctHash.replaceAll(" ", "");
        return correctHash.equals(hash(ctx, pkgname));
    }

    public static String hash(Context ctx, String pkgname) {
        if (pkgname == null) return null;
        try {
            PackageManager pm = ctx.getPackageManager();
            PackageInfo pkginfo =
                pm.getPackageInfo(pkgname, PackageManager.GET_SIGNATURES);
            // Will not handle multiple signatures.
            if (pkginfo.signatures.length != 1) return null;
            Signature sig = pkginfo.signatures[0];
            byte[] cert = sig.toByteArray();
            byte[] sha256 = computeSha256(cert);
            return byte2hex(sha256);
        } catch (NameNotFoundException e) {
            return null;
        }
    }

    private static byte[] computeSha256(byte[] data) {
        try {
            return MessageDigest.getInstance("SHA-256").digest(data);
        } catch (NoSuchAlgorithmException e) {
```

(continues on next page)

(continued from previous page)

```
            return null;
        }
    }

    private static String byte2hex(byte[] data) {
        if (data == null) return null;
        final StringBuilder hexadecimal = new StringBuilder();
        for (final byte b : data) {
            hexadecimal.append(String.format("%02X", b));
        }
        return hexadecimal.toString();
    }
}
```

*** Point 11 *** When generating an APK by [Build] -> [Generate Signed APK], sign the APK with the same developer key that the destination application uses.

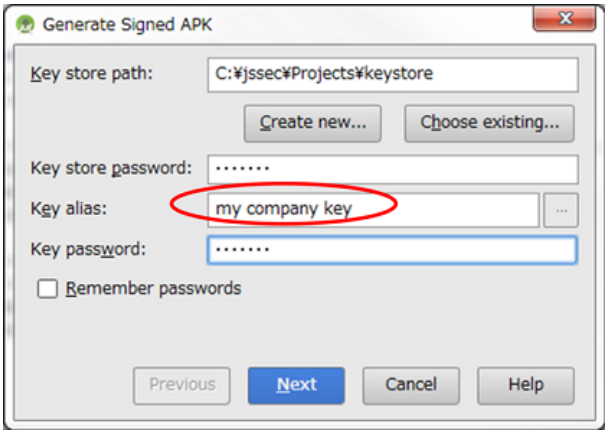


Fig. 5.2.4: Sign the APK with the same developer key that the destination application uses

Signature verification in Android 9.0 (API level 28) and later

APK signature scheme V3 was introduced in Android 9.0 (API level 28) for enabling signature key rotation. At the same time, the package signature-related APIs were also updated[2]. When examining the changes from the standpoint of application signature verification, the hasSigningCertificate() method, which is a new method in thePackageManager class, can now be used for verification. Specifically, this can be substituted for processes such as those where the certificate used for the signature is obtained from the verification target package where the sample code PkgCert class of the Guide was performed and the hash value is calculated. This is applied in the SigPerm and PkgCertWhiteLists in the sample code shown above, and for API level 28 and higher, this new method hasSigningCertificate() is used. Differences in signature schemes and differences in verification as a result of multiple signatures are incorporated into hasSigningCertificate(), and so if targeting API level 28 and higher, use of this is recommended[3].

5.2.1.3 How to Verify the Hash Value of an Application's Certificate

We will provide an explanation on how to verify the hash value of an application's certificate that appears at different points in this Guidebook. Strictly speaking, the hash value means "the SHA256 hash value of the public key certificate for the developer key used to sign the APK."

How to verify it with Keytool

[2] For the specific changes, refer to the Android Developers website (https://developer.android.com/reference/android/content/pm/PackageManager).

[3] As of the time of this writing, there is currently no available Android Support Library compatible with the android.content.pm.PackageManager of Android 9.0 (API level 28).

Using a program called keytool that is bundled with JDK, you can get the hash value (also known as certificate fingerprint) of a public key certificate for the developer key. There are various hash methods such as MD5, SHA1, and SHA256 due to the differences in hash algorithm. However, considering the security strength of the encryption bit length, this Guidebook recommends the use of SHA256. Unfortunately, the keytool bundled to JDK6 that is used in Android SDK does not support SHA256 for calculating hash values. Therefore, it is necessary to use the keytool that is bundled to JDK7 or later.

Example of outputting the content of a debugging certicate of an Android through a keytool

```
> keytool -list -v -keystore <KeystoreFile> -storepass <Password>

Type of keystore: jks
Keystore provider: SUN

One entry is included in a keystore

Other name: androiddebugkey
Date of creation: 2012/05/18
Entry type: PrivateKeyEntry
Length of certificate chain: 1
Certificate[1]:
Owner: CN=Android Debug, O=Android, C=US
Issuer: CN=Android Debug, O=Android, C=US
Serial number: 4fb5d390
Start date of validity period: Fri May 18 13:44:00 JST 2012 End date: Tue Oct 04␣
↪13:44:00 JST 2039
Certificate fingerprint:
         MD5:  8A:1A:E5:15:9A:2A:9A:45:C1:7F:30:EF:17:70:37:D1
         SHA1: 25:BC:25:91:02:A4:DD:04:7D:17:70:EC:41:35:21:00:0C:0A:C7:F1
         SHA256: 0E:FB:72:36:32:83:48:A9:89:71:8B:AD:DF:57:F5:44:D5:CC:B4:AE:B9:DB:
                 34:BC:1E:29:DD:26:F7:7C:82:55
     Signatrue algorithm name: SHA1withRSA
     Subject public key algorithm: 1024-bit RSA key
     Version: 3

*******************************************
*******************************************
```

How to Verify it with JSSEC Certificate Hash Value Checker

Without installing JDK7 or later, you can easily verify the certificate hash value by using JSSEC Certificate Hash Value Checker.

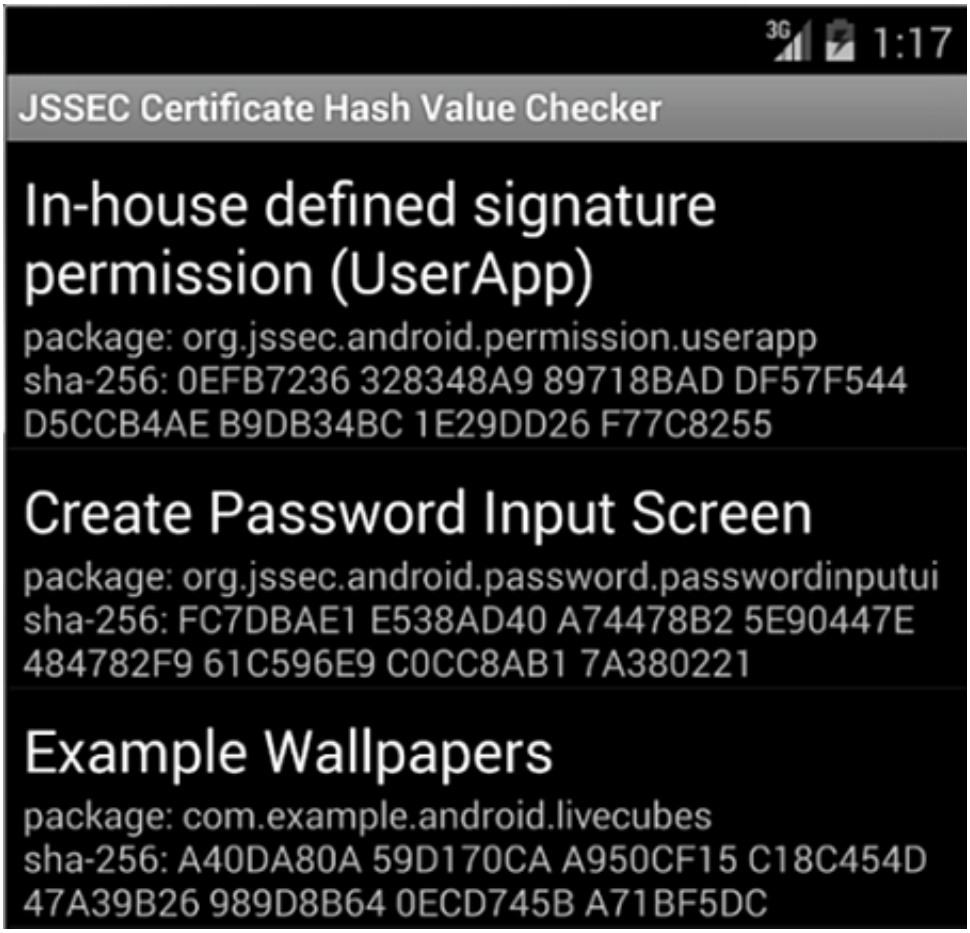


Fig. 5.2.5: JSSEC Certificate Hash Value Checker

This is an Android application that displays a list of certificate hash values of applications which are installed in the device. In the Figure above, the 64-character hexadecimal notation string that is shown on the right of "sha-256" is the certificate hash value. The sample code folder, "JSSEC CertHash Checker" that comes with this Guidebook is the set of source codes. If you would like, you can compile the codes and use it.

5.2.1.4 Methods for using Dangerous Permissions in Android 6.0 and later

Android 6.0 (API Level 23) incorporates modified specifications that are relevant to the implementation of apps---specifically, to the times at which apps are granted permission.

Under the Permission model of Android 5.1 (API Level 22) and earlier versions (See section "5.2.3.6. *Modifications to the Permission model specifications in Android versions 6.0 and later*", all Permissions declared by an app are granted to that app at the time of installation. However, in Android 6.0 and later versions, app developers must explicitly implement apps in such a way that, for Dangerous Permissions, the app requests Permission at appropriate times. When an app requests a Permission, a confirmation window like that shown below is displayed to the Android OS user, requesting a decision from the user as to whether or not to grant the Permission in question. If the user allows the use of the Permission, the app may execute whatever operations require that Permission.

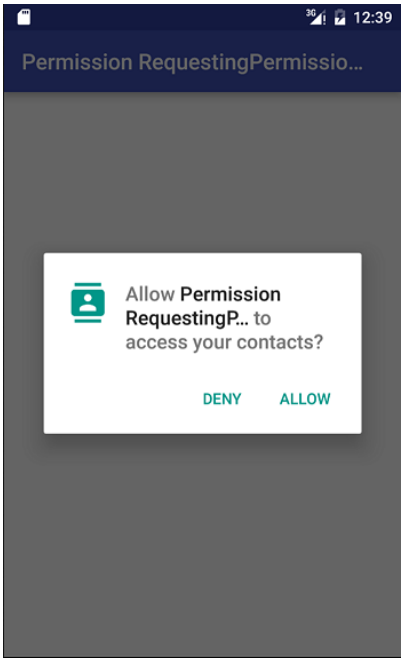


Fig. 5.2.6: Dangerous Permission Confirmation Window

The specifications are also modified regarding the units in which Permissions are granted. Previously, all Permissions were granted simultaneously; in Android 6.0 (API Level 23) and later versions, Permissions are granted by Permission Group. In Android 8.0 (API Level 26) and later versions, Permissions are granted individually. In conjunction with this modification, users are now shown individual confirmation windows for each Permission, allowing users to make more flexible decisions regarding the granting or refusal of Permissions. App developers must revisit the specifications and design of their apps with full consideration paid to the possibility that Permissions may be refused.

For details on the Permission model in Android 6.0 and later, see Section "5.2.3.6. *Modifications to the Permission model specifications in Android versions 6.0 and later*".

Points:

1. Apps declare the Permissions they will use
2. Do not declare the use of unnecessary Permissions
3. Check whether or not Permissions have been granted to the app
4. Request Permissions (open a dialog to request permission from users)
5. Implement appropriate behavior for cases in which the use of a Permission is refused

```
AndroidManifest.xml
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
          >

  <!-- *** POINT 1 *** Apps declare the Permissions they will use -->
  <!-- Permission to read information on contacts (Protection Level: dangerous) -->
  <uses-permission android:name="android.permission.READ_CONTACTS" />

  <!-- *** POINT 2 *** Do not declare the use of unnecessary Permissions -->

  <application
      android:allowBackup="true"
      android:icon="@mipmap/ic_launcher"
      android:label="@string/app_name"
```

(continues on next page)

(continued from previous page)

```
      android:supportsRtl="true"
      android:theme="@style/AppTheme" >
    <activity
        android:name=".MainActivity"
        android:exported="true">
      <intent-filter>
        <action android:name="android.intent.action.MAIN" />

        <category android:name="android.intent.category.LAUNCHER" />
      </intent-filter>
    </activity>
    <activity
        android:name=".ContactListActivity"
        android:exported="false">
    </activity>
  </application>

</manifest>
```

MainActivity.java

```
/*
 * Copyright (C) 2012-2026 Japan Smartphone Security Association
 *
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 *
 *      http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */

package org.jssec.android.permission.permissionrequestingpermissionatruntime;

import android.Manifest;
import android.content.Intent;
import android.content.pm.PackageManager;
import android.os.Bundle;
import androidx.core.app.ActivityCompat;
import androidx.core.content.ContextCompat;
import androidx.appcompat.app.AppCompatActivity;
import android.view.View;
import android.widget.Button;
import android.widget.Toast;

public class MainActivity extends AppCompatActivity
        implements View.OnClickListener {
    private static final int REQUEST_CODE_READ_CONTACTS = 0;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
```

(continues on next page)

(continued from previous page)

```
        setContentView(R.layout.activity_main);

        Button button = (Button)findViewById(R.id.button);
        button.setOnClickListener(this);
    }

    @Override
    public void onClick(View v) {
        readContacts();
    }

    private void readContacts() {
        // *** POINT 3 *** Check whether or not Permissions have been granted to
        // the app
        if (ContextCompat.checkSelfPermission(getApplicationContext(), Manifest.
↪permission.READ_CONTACTS) != PackageManager.PERMISSION_GRANTED) {
            // Permission was not granted
            // *** POINT 4 *** Request Permissions (open a dialog to request
            // permission from users)
            ActivityCompat.requestPermissions(this, new String[]{Manifest.
↪permission.READ_CONTACTS}, REQUEST_CODE_READ_CONTACTS);
        } else {
            // Permission was previously granted
            showContactList();
        }
    }

    // A callback method that receives the result of the user's selection
    @Override
    public void onRequestPermissionsResult(int requestCode, String[] permissions,
                                           int[] grantResults) {
        switch (requestCode) {
        case REQUEST_CODE_READ_CONTACTS:
            if (grantResults.length > 0 &&
                    grantResults[0] == PackageManager.PERMISSION_GRANTED) {
                // Permissions were granted; we may execute operations that use
                // contact information
                showContactList();
            } else {
                // Because the Permission was denied, we may not execute
                // operations that use contact information
                // *** POINT 5 *** Implement appropriate behavior for cases in
                // which the use of a Permission is refused
                Toast.makeText(this,
                               String.format("Use of contact is not allowed."),
                               Toast.LENGTH_LONG).show();
            }
            return;
        }
    }

    // Show contact list
    private void showContactList() {
        // Launch ContactListActivity
        Intent intent = new Intent();
        intent.setClass(getApplicationContext(), ContactListActivity.class);
```

(continues on next page)

(continued from previous page)

```
        startActivity(intent);
    }
}
```

5.2.2 Rule Book

Be sure to follow the rules below when using in-house permission.

1. *System Dangerous Permissions of Android OS Must Only Be Used for Protecting User Assets (Required)*
2. *Your Own Dangerous Permission Must Not Be Used (Required)*
3. *Your Own Signature Permission Must Only Be Defined on the Provider-side Application (Required)*
4. *Verify If the In-house-defined Signature Permission Is Defined by an In-house Application (Required)*
5. *Your Own Normal Permission Should Not Be Used (Recommended)*
6. *The String for Your Own Permission Name Should Be of an Extent of the Package Name of Application (Recommended)*

5.2.2.1 System Dangerous Permissions of Android OS Must Only Be Used for Protecting User Assets (Required)

Since the use of your own dangerous permission is not recommended (please refer to "5.2.2.2. *Your Own Dangerous Permission Must Not Be Used (Required)*", we will proceed on the premise of using system dangerous permission of Android OS.

Unlike the other three types of permissions, dangerous permission has a feature that requires the user's consent to the grant of the permission to the application. When installing an application on a device that has declared a dangerous permission to use, the following screen will be displayed. Subsequently, the user is able to know what level of permission (dangerous permission and normal permission) the application is trying to use. When the user taps "install", the application will be granted the permission and then it will be installed.

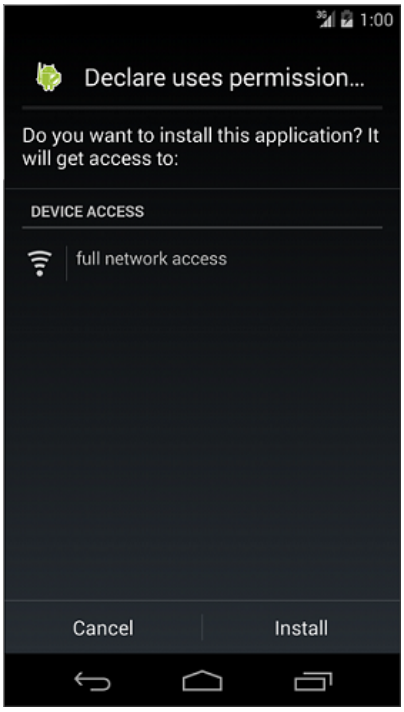


Fig. 5.2.7: System Dangerous Permissions of Android OS Confirmation Window

An application can handle user assets and assets that the developer wants to protect. We must be aware that dangerous permission can protect only user assets because the user is just who the granting of permission is entrusted to. On the other hand, assets that the developer wants to protect cannot be protected by the method above.

For example, suppose that an application has a Component that communicates only with an In-house application, it doesn't permit the access to the Component from any applications of the other companies, and it is implemented that it's protected by dangerous permission. When a user grants permission to an application of another company based on the user's judgment, in-house assets that need to be protected may be exploited by the application granted. In order to provide protection for in-house assets in such cases, we recommend the usage of in-house-defined signature permission.

5.2.2.2 Your Own Dangerous Permission Must Not Be Used (Required)

Even when in-house-defined Dangerous Permission is used, the screen prompt "Asking for the Allowance of Permission from User" is not displayed in some cases. This means that at times the feature that asks for permission based on the judgment of a user, which is the characteristic of Dangerous Permission, does not function. Accordingly, the Guidebook will make the rule "In-house -defined dangerous permission must not be used".

In order to explain it, we assume two types of applications. The first type of application defines an in-house dangerous permission, and it is an application that makes a Component, which is protected by this permission, public. We call this ProtectedApp. The other is another application which we call AttackerApp and it tries to exploit the Component of ProtectedApp. Also we assume that the AttackerApp not only declares the permission to use it, but also defines the same permission.

AttackerApp can use the Component of a ProtectedApp without the consent of a user in the following cases:

1. When the user installs the AttackerApp, the installation will be completed without the screen prompt that asks for the user to grant the application the dangerous permission.
2. Similarly, when the user installs the ProtectedApp, the installation will be completed without any special warnings.
3. When the user launches the AttackerApp afterwards, the AttackerApp can access the Component of the ProtectedApp without being detected by the user, which can potentially lead to damage.

The cause of this case is explained in the following. When the user tries to install the AttackerApp first, the permission that has been declared for usage with uses-permission is not defined on the particular device yet. Finding no error, Android OS will continue the installation. Since the user consent for dangerous permission is required only at the time of installation, an application that has already been installed will be handled as if it has been granted permission. Accordingly, if the Component of an application which is installed later is protected with the dangerous permission of the same name, the application which was installed beforehand without the user permission will be able to exploit the Component.

Furthermore, since the existence of system dangerous permissions defined by Android OS is guaranteed when an application is installed, the user verification prompt will be displayed every time an application with uses-permission is installed. This problem arises only in the case of self-defined dangerous permission.

At the time of this writing, no viable method to protect the access to the Component in such cases has been developed yet. Therefore, your own dangerous permission must not be used.

5.2.2.3 Your Own Signature Permission Must Only Be Defined on the Provider-side Application (Required)

As demonstrated in, "5.2.1.2. *How to Communicate Between In-house Applications with In-house-defined Signature Permission*", the security can be assured by checking the signature permission at the time of executing intercommunications between In-house applications. When using this mechanism, the definition of the permission whose Protection Level is signature must be written in AndroidManifest.xml of the provider-side application that has the Component, but the user-side application must not define the signature permission.

This rule is applied to signatureOrSystem Permission as well.

The reason for this is as follows.

We assume that there are multiple user-side applications that have been installed prior to the provider-side application and every user-side application not only has required the signature permission that the provider-side application has defined, but also has defined the same permission. Under these circumstances, all user-side applications will be able to access the provider-side application just after the provider-side application is installed. Subsequently, when the user-side application that was installed first is uninstalled, the definition of the permission also will be deleted and then the permission will turn out to be undefined. As a result, the remaining user-side applications will be unable to access to the provider-side application.

In this manner, when the user-side application defines a self-defined permission, it can unexpectedly turn out the permission to be undefined. Therefore, only the provider-side application providing the Component that needs to be protected should define the permission, and defining the permission on the user-side must be avoided.

By doing as mentioned just above, the self-defined permission will be applied by Android OS at the time of the installation of the provider-side application, and the permission will turn out to be undefined at the time of the uninstallation of the application. Therefore, since the existence of the permission's definition always corresponds to that of the provider-side application, it is possible to provide an appropriate Component and protect it. Please be aware that this argument stands because regarding in-house-defined signature permission the user-side application is granted the permission regardless of the installation order of applications in inter-communication[4].

5.2.2.4 Verify If the In-house-defined Signature Permission Is Defined by an In-house Application (Required)

Actuality, you cannot say to be secure enough only by declaring a signature permission through AnroidManifest.xml and protecting the Component with the permission. For the details of this issue, please refer to, "5.2.3.1. *Characteristics of Android OS that Avoids Self-defined Signature Permission and Its Counter-measures*" in the Advanced Topics section.

The following are the steps for using in-house-defined signature permission securely and correctly.

First, write as the followings in AndroidManifest.xml:

1. Define an in-house signature permission in the AndroidManifest.xml of the provider-side application. (definition of permission)
 Example: <permission android:name="xxx" android:protectionLevel="signature" />
2. Enforce the permission with the permission attribute of the Component to be protected in the AndroidManifest.xml of the provider-side application. (enforcement of permission)
 Example: <activity android:permission="xxx" ... >...</activity>
3. Declare the in-house-defined signature permission with the uses-permission tag in the AndroidManifest.xml of every user-side application to access the Component to be protected. (declaration of using permission)
 Example: <uses-permission android:name="xxx" />

Next, implement the followings in the source code.

4. Before processing a request to the Component, first verify that the in-house-defined signature permission has been defined by an in-house application. If not, ignore the request. (protection in the provider-side component)
5. Before accessing the Component, first verify that the in-house-defined signature permission has been defined by an in-house application. If not, do not access the Component (protection in the user-side component).

Lastly, execute the following with the Signing function of Android Studio.

6. Sign APKs of all inter-communicating applications with the same developer key.

Here, for specific points on how to implement "Verify that the in-house-defined signature permission has been defined by an In house application", please refer to "5.2.1.2. *How to Communicate Between In-house Applications with In-house-defined Signature Permission*".

This rule is applied to signatureOrSystem Permission as well.

[4] If using normal/dangerous permission, the permission will not be granted the user-side application if the user-side application is installed before the provider-side application, the permission remains undefined. Therefore, the Component cannot be accessed even after the provider-side application has been installed.

5.2.2.5 Your Own Normal Permission Should Not Be Used (Recommended)

An application can use a normal permission just by declaring it with uses-permission in AndroidManifest.xml. Therefore, you cannot use a normal permission for the purpose of protecting a Component from a malware installed.

Furthermore, in the case of inter-application communication with self-defined normal permission, whether an application can be granted the permission depends on the order of installation. For example, when you install an application (user-side) that has declared to use a normal permission prior to another application (provider-side) that possesses a Component which has defined the permission, the user-side application will not be able to access the Component protected with the permission even if the provider-side application is installed later.

As a way to prevent the loss of inter-application communication due to the order of installation, you may think of defining the permission in every application in the communication. By this way, even if a user-side application has been installed prior to the provider-side application, all user-side applications will be able to access the provider-side application. However, it will create a situation that the permission is undefined when the user-side application installed first is uninstalled. As a result, even if there are other user-side applications, they will not be able to gain access to the provider-side application.

As stated above, there is a concern of damaging the availability of an application, thus your own normal permission should not be used.

5.2.2.6 The String for Your Own Permission Name Should Be of an Extent of the Package Name of Application (Recommended)

When multiple applications define permissions under the same name, the Protection Level that has been defined by an application installed first will be applied. Protection by signature permission will not be available in the case that the application installed first defines a normal permission and the application installed later defines a signature permission under the same name. Even in the absence of malicious intent, a conflict of permission names among multiple applications could cause behavior s of any applications as an unintended Protection Level. To prevent such accidents, it is recommended that a permission name extends (starts with) the package name of the application defining the permission as below.

(package name).permission.(identifying string)

For example, the following name would be preferred when defining a permission of READ access for the package of org.jssec.android.sample.

org.jssec.android.sample.permission.READ

5.2.3 Advanced Topics

5.2.3.1 Characteristics of Android OS that Avoids Self-defined Signature Permission and Its Counter-measures

Self-defined signature permission is a permission that actualizes inter-application communication between the applications signed with the same developer key. Since a developer key is a private key and must not be public, there is a tendency to use signature permission for protection only in cases where in-house applications communicate with each other.

First, we will describe the basic usage of self-defined signature permission that is explained in the Developer Guide ([https://developer.android.com/guide/topics/security/security.html{]}(https://developer.android.com/guide/topics/security/security.html)) of Android. However, as it will be explained later, there are problems with regard to the avoidance of permission. Consequently, counter-measures that are described in this Guidebook are necessary.

The followings are the basic usage of self-defined Signature Permission.

1. Define a self-defined signature permission in the AndroidManifest.xml of the provider-side application. (definition of permission)
 Example: <permission android:name="xxx" android:protectionLevel="signature" />

2. Enforce the permission with the permission attribute of the Component to be protected in the AndroidManifest.xml of the provider-side application. (enforcement of permission)
 Example: <activity android:permission="xxx" ... >...</activity>
3. Declare the self-defined signature permission with the uses-permission tag in the AndroidManifest.xml of every user-side application to access the Component to be protected. (declaration of using permission)
 Example: <uses-permission android:name="xxx" />
4. Sign APKs of all inter-communicating applications with the same developer key.

Actually, if the following conditions are fulfilled, this approach will create a loophole to avoid signature permission from being performed.

For the sake of explanation, we call an application that is protected by self-defined signature permission as ProtectedApp, and AttackerApp for an application that has been signed by a different developer key from the ProtectedApp. What a loophole to avoid signature permission from being performed means is, despite the mismatch of the signature for AttackerApp, it is possible to gain access to the Component of ProtectedApp.

1. An AttackerApp also defines a normal permission (strictly speaking, signature permission is also acceptable) under the same name as the signature permission which has been defined by the ProtectedApp.
 Example: <permission android:name=" xxx" android:protectionLevel="normal" />
2. The AttackerApp declares the self-defined normal permission with uses-permission.
 Example: <uses-permission android:name="xxx" />
3. The AttackerApp has installed on the device prior to the ProtectedApp.

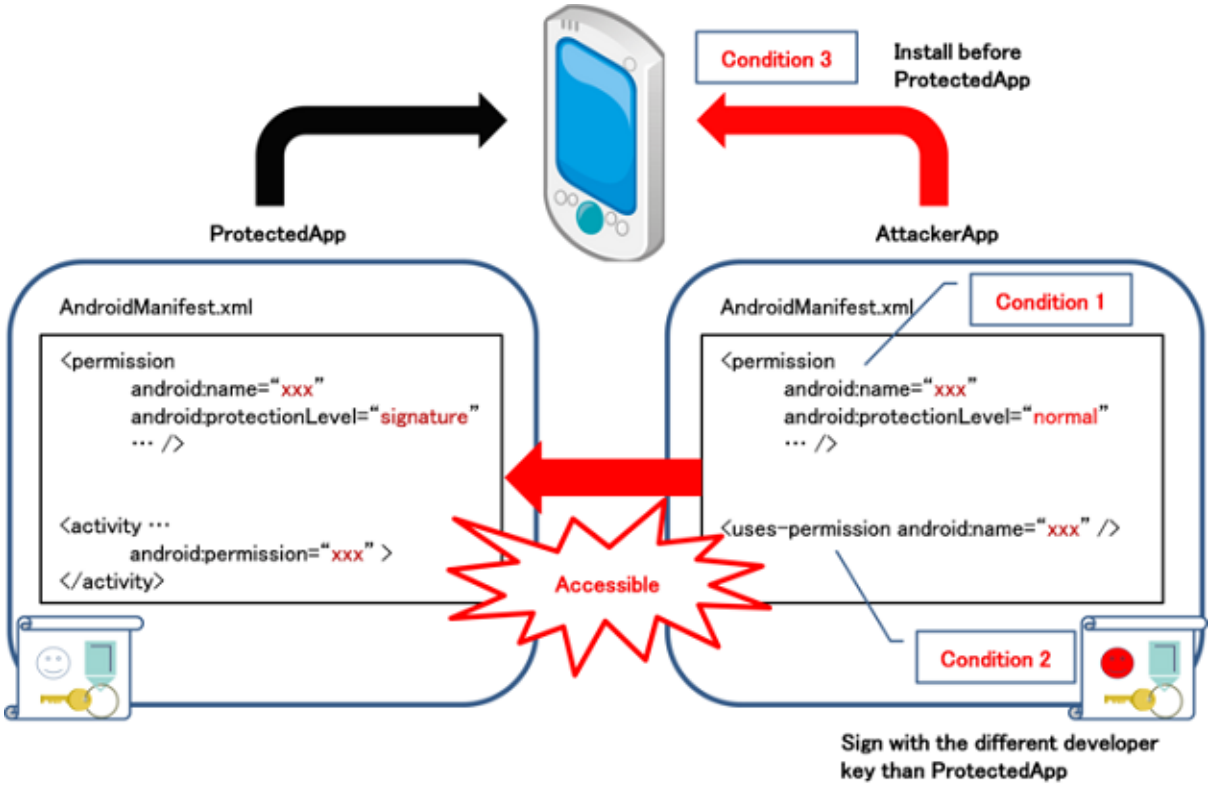


Fig. 5.2.8: A loophole to avoid Signature Permission

The permission name that is necessary to meet Condition 1 and Condition 2 can easily be known by an attacker taking AndroidManifest.xml out from an APK file. The attacker also could satisfy Condition 3 with a certain amount of effort (e.g. deceiving a user).

There is a risk of self-defined signature permission to evade protection if only the basic usage is adopted, and a counter-measure to prevent such loopholes is needed. Specifically, you could find how to solve the above-mentioned

issues by using the method described in "5.2.2.4. *Verify If the In-house-defined Signature Permission Is Defined by an In-house Application (Required)*".

5.2.3.2 Falsification of AndroidManifest.xml by a User

We have already touched on the case that a Protection Level of self-defined permission could be changed as not intended. To prevent malfunctioning due to such cases, it has been needed to implement some sort of counter-measures on the source-code side of Java. From the viewpoint of AndroidManifest.xml falsification, we will talk about the counter-measures to be taken on the source-code side. We will demonstrate a simple case of installation that can detect falsifications. However, please note that these counter-measures are little effective against professional hackers who falsify with criminal intent.

This section is about the falsification of an application and users with malicious intent. Although this is originally outside of the scope of a Guidebook, from the fact that this is related to Permission and the tools for such falsification are provided in public as Android applications, we decided to mention it as "Simple counter-measures against amateur hackers".

It must be remembered that applications that can be installed from market are applications that can be falsified without root privilege. The reason is that applications that can rebuild and sign APK files with altered AndroidManifest.xml are distributed. By using these applications, anyone can delete any permission from applications they have installed.

As an example, there seems to be cases of rebuilding APKs with different signatures altering AndroidManifest.xml with INTERNET permission removed to render advertising modules attached in applications as useless. There are some users who praise these types of tools due to the fact that no personal information is leaked anywhere. As these ads which are attached in applications stop functioning, such actions cause monetary damage for developers who are counting on ad revenue. And it is believed that most of the users don't have any compunction.

In the following code, we show an instance of implementation that an application that has declared INTERNET permission with uses-permission verifies if INTERNET permission is described in the AndroidManifest.xml of itself at run time.

```
public class CheckPermissionActivity extends Activity {

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);

        // Acquire Permission defined in AndroidManifest.xml
        List<String> list = getDefinedPermissionList();

        // Detect falsification
        if( checkPermissions(list) ){
            // OK
            Log.d("dbg", "OK.");
        }else{
            Log.d("dbg", "manifest file is stale.");
            finish();
        }
    }

    /**
     * Acquire Permission through list that was defined in AndroidManifest.xml
     * @return
     */
    private List<String> getDefinedPermissionList(){
        List<String> list = new ArrayList<String>();
        list.add("android.permission.INTERNET");
        return list;
```

(continues on next page)

(continued from previous page)

```
    }

    /**
     * Verify that Permission has not been changed Permission
     * @param permissionList
     * @return
     */
    private boolean checkPermissions(List<String> permissionList){
        try {
            PackageInfo packageInfo = getPackageManager().getPackageInfo(
                    getPackageName(), PackageManager.GET_PERMISSIONS);
            String[] permissionArray = packageInfo.requestedPermissions;
            if (permissionArray != null) {
                for (String permission : permissionArray) {
                    if(! permissionList.remove(permission)){
                        // Unintended Permission has been added
                        return false;
                    }
                }
            }

            if(permissionList.size() == 0){
                // OK
                return true;
            }

        } catch (NameNotFoundException e) {
        }

        return false;
    }
}
```

5.2.3.3 Detection of APK Falsification

We explained about detecting the falsification of permissions by a user in "5.2.3.2. *Falsification of AndroidManifest.xml by a User*". However, the falsification of applications is not limited to permission only, and there are many other cases where applications are appropriated without any changes in the source code. For example, it is a case where they distribute other developers' applications (falsified) in the market as if they were their own applications just by replacing resources to their own. Here, we will show a more generic method to detect the falsification of an APK file.

In order to falsify an APK, it is needed to decode the APK file into folders and files, modify their contents, and then rebuild them into a new APK file. Since the falsifier does not have the key of the original developer, he would have to sign the new APK file with his own key. As the falsification of an APK inevitably brings with a change in signature (certificate), it is possible to detect whether an APK has been falsified at run time by comparing the certificate in the APK and the developer's certificate embedded in the source code as below.

The following is a sample code. Also, a professional hacker will be able to easily circumvent the detection of falsification if this implementation example is used as it is. Please apply this sample code to your application by being aware that this is a simple implementation example.

Points:

1. Verify that an application's certificate belongs to the developer before major processing is started.

SignatureCheckActivity.java

```
/*
 * Copyright (C) 2012-2026 Japan Smartphone Security Association
 *
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 *
 *      http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */

package org.jssec.android.permission.signcheckactivity;

import org.jssec.android.shared.PkgCert;
import org.jssec.android.shared.Utils;

import android.app.Activity;
import android.content.Context;
import android.os.Bundle;
import android.widget.Toast;

public class SignatureCheckActivity extends Activity {
    // Self signed certificate hash value
    private static String sMyCertHash = null;
    private static String myCertHash(Context context) {
        if (sMyCertHash == null) {
            if (Utils.isDebuggable(context)) {
                // Certificate hash value of "androiddebugkey" of
                // debug.keystore
                sMyCertHash = "0EFB7236 328348A9 89718BAD DF57F544 D5CCB4AE␣
↪B9DB34BC 1E29DD26 F77C8255";
            } else {
                // Certificate hash value of "my company key" of keystore
                sMyCertHash = "D397D343 A5CBC10F 4EDDEB7C A10062DE 5690984F␣
↪1FB9E88B D7B3A7C2 42E142CA";
            }
        }
        return sMyCertHash;
    }

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);

        // *** POINT 1 *** Verify that an application's certificate belongs to the
        // developer before major processing is started
        if (!PkgCert.test(this, this.getPackageName(), myCertHash(this))) {
            Toast.makeText(this, "Self-sign match  NG", Toast.LENGTH_LONG).show();
            finish();
            return;
```

(continues on next page)

(continued from previous page)

```
        }
        Toast.makeText(this, "Self-sign match  OK", Toast.LENGTH_LONG).show();
    }
}
```

PkgCert.java

```
/*
 * Copyright (C) 2012-2026 Japan Smartphone Security Association
 *
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 *
 *      http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */

package org.jssec.android.shared;

import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;

import android.content.Context;
import android.content.pm.PackageInfo;
import android.content.pm.PackageManager;
import android.content.pm.PackageManager.NameNotFoundException;
import android.content.pm.Signature;

public class PkgCert {

    public static boolean test(Context ctx, String pkgname, String correctHash) {
        if (correctHash == null) return false;
        correctHash = correctHash.replaceAll(" ", "");
        return correctHash.equals(hash(ctx, pkgname));
    }

    public static String hash(Context ctx, String pkgname) {
        if (pkgname == null) return null;
        try {
            PackageManager pm = ctx.getPackageManager();
            PackageInfo pkginfo =
                pm.getPackageInfo(pkgname, PackageManager.GET_SIGNATURES);
            // Will not handle multiple signatures.
            if (pkginfo.signatures.length != 1) return null;
            Signature sig = pkginfo.signatures[0];
            byte[] cert = sig.toByteArray();
            byte[] sha256 = computeSha256(cert);
            return byte2hex(sha256);
        } catch (NameNotFoundException e) {
            return null;
        }
```

(continues on next page)

(continued from previous page)

```
    }

    private static byte[] computeSha256(byte[] data) {
        try {
            return MessageDigest.getInstance("SHA-256").digest(data);
        } catch (NoSuchAlgorithmException e) {
            return null;
        }
    }

    private static String byte2hex(byte[] data) {
        if (data == null) return null;
        final StringBuilder hexadecimal = new StringBuilder();
        for (final byte b : data) {
            hexadecimal.append(String.format("%02X", b));
        }
        return hexadecimal.toString();
    }
}
```

5.2.3.4 Permission Re-delegation Problem

An application must declare to use permission when accessing contacts or GPS with its information and features that are protected by Android OS. When the permission required is granted, the permission is delegated to the application and the application would be able to access the information and features protected with the permission.

Depending on how the program is designed, the application to which has been delegated (granted) the permission is able to acquire data that is protected with the permission. Furthermore, the application can offer another application the protected data without enforcing the same permission. This is nothing less than permission-less application to access data that is protected by permission. This is virtually the same thing as re-delegating the permission, and this is referred to the Permission Re-delegation Problem. Accordingly, the specification of the permission mechanism of Android only is able to manage permission of direct access from an application to protected data.

A specific example is shown in Fig. 5.2.9. The application in the center shows that an application which has declared android.permission.READ_CONTACTS to use it reads contacts and then stores them into its own database. The Permission Re-delegation Problem occurs when information that has been stored is offered to another application without any restriction via Content Provider.

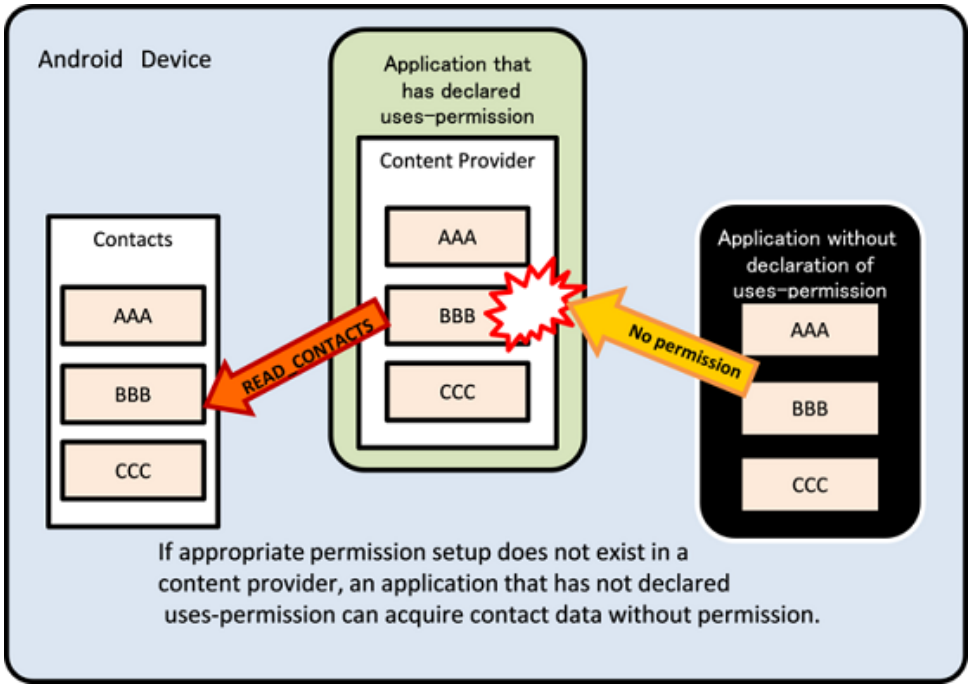


Fig. 5.2.9: An Application without Permission Acquires Contacts

As a similar example, an application that has declared android.permission.CALL_PHONE to use it receives a phone number (maybe input by a user) from another application that has not declared the same permission. If that number is being called without the verification of a user, then also there is the Permission Re-delegation Problem.

There are cases where the secondary provision of another application with nearly-intact information asset or functional asset acquired with the permission is needed. In those cases, the provider-side application must demand the same permission for the provision in order to maintain the original level of protection. Also, in the case of only providing a portion of information asset as well as functional asset in a secondary fashion, an appropriate amount of protection is necessary in accordance with the degree of damage that is incurred when a portion of that information or functional asset is exploited. We can use protective measures such as demanding permission as similar to the former, verifying user consent, and setting up restrictions for target applications by using "4.1.1.1. *Creating/Using Private Activities*", or "4.1.1.4. *Creating/Using In-house Activities*" etc.

Such Permission Re-delegation Problem is not only limited to the issue of the Android permission. For an Android application, it is generic that the application acquires necessary information/functions from different applications, networks, and storage media. And in many cases, some permissions as well as restrictions are needed to access them. For example, if the provider source is an Android application, it is the permission, if it is a network, then it is the log-in, and if it is a storage media, there will be access restrictions. Therefore, such measures need to be implemented for an application after carefully considering as information/functions are not used in the contrary manner of the user's intention. This is especially important at the time of providing acquired information/functions to another application in a secondary manner or transferring to networks or storage media. Depending on the necessity, you have to enforce permission or restrict usage like the Android permission. Asking for the user's consent is part of the solution.

In the following code, we demonstrate a case where an application that acquires a list from the contact database by using READ_CONTACTS permission enforces the same READ_CONTACTS permission on the information destination source.

Point

1. Enforce the same permission that the provider does.

```
AndroidManifest.xml
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
          package="org.jssec.android.permission.transferpermission" >
```

(continues on next page)

(continued from previous page)

```
  <uses-permission android:name="android.permission.READ_CONTACTS"/>

  <application
      android:allowBackup="false"
      android:icon="@drawable/ic_launcher"
      android:label="@string/app_name"
      android:theme="@style/AppTheme" >
    <activity
        android:name=".TransferPermissionActivity"
        android:label="@string/title_activity_transfer_permission" >
      <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER" />
      </intent-filter>
    </activity>

    <!-- *** Point1 *** Enforce the same permission that the rovider does. -->
    <provider
        android:name=".TransferPermissionContentProvider"
        android:authorities="org.jssec.android.permission.transferpermission"
        android:enabled="true"
        android:exported="true"
        android:readPermission="android.permission.READ_CONTACTS" >
    </provider>
  </application>

</manifest>
```

When an application enforces multiple permissions, the above method will not solve it. By using Context#checkCallingPermission() or PackageManager#checkPermission() from the source code, verify whether the invoker application has declared all permissions with uses-permission in the Manifest.

In the case of an Activity

```
public void onCreate(Bundle savedInstanceState) {
    [...]
    if (checkCallingPermission("android.permission.READ_CONTACTS") ==
        PackageManager.PERMISSION_GRANTED
     && checkCallingPermission("android.permission.WRITE_CONTACTS") ==
        PackageManager.PERMISSION_GRANTED) {
            // Processing during the time when an invoker is correctly
        // declaring to use
            return;
    }
    finish();
}
```

5.2.3.5 Signature check mechanism for custom permissions (Android 5.0 and later)

In versions of Android 5.0 (API Level 21) and later, the application which defines its own custom permissions cannot be installed if the following conditions are met.

1. Another application which defines its own permission with the same name has already installed on the device.
2. The applications are signed with different keys

When both an application with the protected function (Component) and an application using the function define their own permission with the same name and are signed with the same key, the above mechanism will protect against installation of other company's applications which define their own custom permission with the same name. However, as mentioned in "5.2.2.3. *Your Own Signature Permission Must Only Be Defined on the Provider-side Application (Required)*", that mechanism won't work well for checking if a custom permission is defined by your own company because the permission could be undefined without your intent by uninstalling applications when plural applications define the same permission.

To sum it up, also in versions of Android 5.0 (API Level 21) and later, you are required to comply with the two rules, "5.2.2.3. *Your Own Signature Permission Must Only Be Defined on the Provider-side Application (Required)*" and "5.2.2.4. *Verify If the In-house-defined Signature Permission Is Defined by an In-house Application (Required)*" when your application defines your own Signature Permission.

5.2.3.6 Modifications to the Permission model specifications in Android versions 6.0 and later

Android 6.0 (API Level 23) introduces modified specifications for the Permission model that affect both the design and specifications of apps. In this section we offer an overview of the Permission model in Android 6.0 and later. We also describe modifications made in Android 8.0 and later as well as “one-time permission” of Android 11.0 and later.

The timing of permission grants and refusals

In cases where an app declares use of permissions requiring user confirmation (Dangerous Permissions) [see Section “5.2.2.1. *System Dangerous Permissions of Android OS Must Only Be Used for Protecting User Assets (Required)*”], the specifications for Android 5.1 (API level 22) and earlier versions called for a list of such permissions to be displayed when the app is installed, and the user must grant all permissions for the installation to proceed. At this point, all permissions declared by the app (including permissions other than Dangerous Permissions) were granted to the app; once these permissions were granted to the app, they remained in effect until the app was uninstalled from the terminal.

However, in the specifications for Android 6.0 and later versions, the granting of permissions takes place when an app is executed. The granting of permissions, and user confirmation of permissions, does not take place when the app is installed. When an app executes a procedure that requires Dangerous Permissions, it is necessary to check whether or not those permissions have been granted to the app in advance; if not, a confirmation window must be displayed in Android OS to request permission from the user[5]. If the user grants permission from the confirmation window, the permissions are granted to the app. However, permissions granted to an app by a user (Dangerous Permissions) may be revoked at any time via the Settings menu (Fig. 5.2.10). For this reason, appropriate procedures must be implemented to ensure that apps cause no irregular behavior even in situations in which they cannot access needed information or functionality because permission has not been granted.

[5] Because Normal Permissions and Signature Permissions are automatically granted by Android OS, there is no need to obtain user confirmation for these permissions.

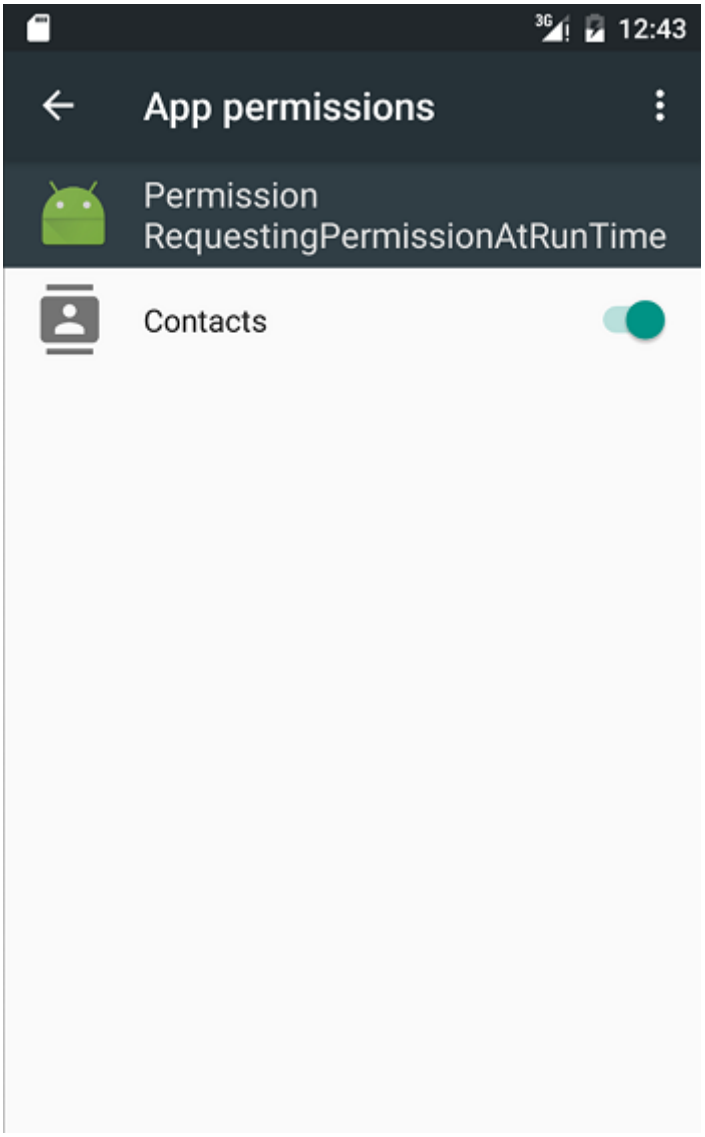


Fig. 5.2.10: App Permissions Window

Also, from Android 11.0, an option “only this time” is available for selection (one-time permission) if granting permission when executing some permissions related to location information, microphone, and camera. This permission is used to enable permissions only while apps are executed and the permission will be invalid when the app is closed or when a certain time[6] passes after moving to the background. Special granting measures are not required if granting of permissions has been implemented on Android 6.0 or later.

- android.permission.ACCESS_FINE_LOCATION
- android.permission.ACCESS_BACKGROUND_LOCATION
- android.permission.RECORD_AUDIO
- android.permission.CAMERA

Units of permission grants and refusals

Multiple Permissions may be grouped together into what is known as a Permission Group based on their functions and type of information relevant to them. For example, the Permission android.permission.READ_CALENDAR, which is required to read calendar information, and the Permission android.permission.WRITE_CALENDAR, which

[6] Was 1 minute on emulators and the actual device (Pixel 3) when confirmed on Android 11. However, this may differ depending on the terminal.

is required to write calendar information, are both affiliated with the Permission Group named android.permission-group.CALENDAR.

In the Permission model for Android 6.0 (API Level 23) and later, privileges are granted or denied at the block-unit level of the Permission Group, as shown here. However, developers must be careful to note that the block unit may vary depending on the combination of OS and SDK (see below).

- For terminals running Android 6.0 (API Level 23) or later and app targetSdkVersion: 23~25

If android.permission.READ_CALENDAR and android.permission.WRITE_CALENDAR are listed in the Manifest, then when the app is launched a request for android.permission.READ_CALENDAR is issued; if the user grants this permission, Android OS determines that both android.permission.READ_CALENDAR and android.permission.WRITE_CALENDAR are permitted for use and thus grants the permission.

- For terminals running Android 8.0 (API Level 26 or later and app targetSdkVersion 26 and above:

Only requested Permissions are granted. Thus, even if android.permission.READ_CALENDAR and android.permission.WRITE_CALENDAR are both listed, if only android.permission.READ_CALENDAR has been requested and granted by the user, then only android.permission.READ_CALENDAR will be granted. Thereafter, if android.permission.WRITE_CALENDAR is requested, the permission will be granted immediately with no dialog box shown to the user[7].

Also, in contrast to the granting of permissions, cancelling of permissions from the settings menu is carried out at the block-unit level of the Permission Group on Android 8.0 or later.

For more information on the classification of Permission Groups, see the Developer Reference ([https://developer.android.com/guide/topics/permissions/overview#perm-groups{]}(https://developer.android.com/guide/topics/permissions/overview#perm-groups)).

The affected range of the revised specifications

Cases in which apps require Permission requests at runtime are restricted to situations in which the terminal is running Android 6.0 or later and the app's targetSDKVersion is 23 or higher. If the terminal is running Android 5.1 or earlier, or if the app's targetSDKVersion was 23 or lower, permissions are requested and granted altogether at the time of installation, as was traditionally the case. However, if the terminal is running Android 6.0 or later, then—even if the app's targetSDKVersion is below 23—permissions that were granted by the user at installation may be revoked by the user at any time. This creates the possibility of unintended irregular app termination. Developers must either comply immediately with the modified specifications or set the maxSDKVersion of their app to 22 or earlier to ensure that the app cannot be installed on terminals running Android 6.0 (API Level 23) or later.

Furthermore, in devices running Android 10 or later, when an app targeting devices running Android 5.1 (API Level 22) or earlier is executed for the first time, a warning is displayed indicating that it may not run properly. Also, for apps that request granting of storage access and other permissions by the user, a permission (Allow/Deny) selection screen appears before this warning[8].

Table 5.2.1: Timing at which app is granted permissions

Terminal Android OS Version	App targetSDKVersion	Timing at which app is granted permissions | User has control over permissions	
>=8.0	>=26	App execution (granted individually)	Yes
	<26	App execution (granted by Permission Group)	Yes
	<23	App installation	Yes (rapid response required)
>=6.0	>=23	App execution (granted by Permission Group)	Yes
	<23	App installation	Yes (rapid response required)
<=5.1	>=23	App installation	No
	<23	App installation	No

[7] In this case as well, the app must declare usage of both android.permission.READ_CALENDAR and android.permission.WRITE_CALENDAR.

[8] https://developer.android.com/about/versions/10/behavior-changes-all#low-target-sdk-warnings

However, it should be noted that the effect of maxSdkVersion is limited. When the value of maxSdkVersion is set 22 or earlier, Android 6.0 (API Level 23) and later of the devices are no longer listed as an installable device of the target application in Google Play. On the other hand, because the value of maxSdkVersion is not checked in the marketplace other than Google Play, it may be possible to install the target application in the Android 6.0 (API Level 23) or later.

Because the effect of maxSdkVersion is limited, and further Google does not recommend the use of maxSdkVersion, it is recommended that developers comply immediately with the modified specifications.

In Android 6.0 and later versions, permissions for the following network communications have their Protection Level changed from Dangerous to Normal. Thus, even if apps declare the use of these Permissions, there is no need to acquire explicit permission from the user, and hence the modified specification has no impact in this case.

- android.permission.BLUETOOTH
- android.permission.BLUETOOTH_ADMIN
- android.permission.CHANGE_WIFI_MULTICAST_STATE
- android.permission.CHANGE_WIFI_STATE
- android.permission.CHANGE_WIMAX_STATE
- android.permission.DISABLE_KEYGUARD
- android.permission.INTERNET
- android.permission.NFC

5.2.3.7 Function That Automatically Resets Unused App Permissions in Android 11.0 and Later

A function that resets permissions of apps that have not been used for a certain period of time in Android 11.0 (API Level 30) has been added. The default reset setting varies depending on targetSDKVersion. This function is set by turning on the “Remove permissions if app isn’t used” option on the app permission setting screen.

- targetSDKVersion=30: Enabled in the default state
- targetSDKVersion<30: Disabled in the default state

The target permissions are those with Protection Levels at Dangerous Permission[9]. However, permissions that enable access once will be set so that confirmations will be required each time if once is selected.

When using functions that require permission, errors do not occur if confirming permissions each time. However, there may be cases where apps that are always running in the background may stop while unnoticed. For this reason, on these apps, it is necessary to request users to disable auto-reset.

1. Confirm that the auto-reset function is disabled using PackageManager#isAutoRevokeWhitelisted() (returns true if disabled).
2. If the auto-reset function is enabled, call out the app setting screen, and guide users to the permissions setting screen.

However, isAutoRevokeWhitelisted() is an API added at API Level 30 and cannot be determined by the app with targetSDKVersion lower than 30. For this reason, apps that have targetSDKVersion lower than 30 and that are required to have auto-reset disabled need to have a flow to review the auto-reset setting in the case where the auto-reset setting was changed by users, in addition to demand for application of permission again if the permission is canceled.

[9] Permission remained granted even though auto-reset was performed for ACTIVITY_RECOGNITION (physical activity) when confirmed with Android 11.

5.2.3.8 Revoking Runtime Permissions

Starting from Android 13, apps can revoke access to runtime permissions granted by the user by using the following APIs.

- revokeSelfPermissionOnKill()
- revokeSelfPermissionsOnKill()

To revoke a specific runtime permission, specify the name of that permission using revokeSelfPermissionOnKill(). To specify multiple runtime permissions at once, specify the names of the permissions together in revokeSelfPermissionsOnKill(). If no permission was granted, the APIs will do nothing.

The introduction of these APIs follows the principle of minimal permissions, and it is recommended that runtime permissions be reviewed periodically by the app and unused permissions be revoked.

The sample code is as follows. This shows an example of an app that uses android.permission.CAMERA and android.permission.CALL_PHONE permissions.

First, this is an example of the declaration.

```
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE">
<uses-permission android:name="android.permission.CALL_PHONE" />
```

Next, the following is an example of code for requesting or revoking two permissions when a button is pressed.

```
private final String[] PERMISSIONS = {
        Manifest.permission.CAMERA,
        Manifest.permission.CALL_PHONE
};
private ActivityResultContracts.RequestMultiplePermissions␣
↪multiplePermissionsContract;
private ActivityResultLauncher<String[]> multiplePermissionLauncher;

@RequiresApi(api = 33)
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);

    multiplePermissionsContract = new ActivityResultContracts.
↪RequestMultiplePermissions();
    multiplePermissionLauncher =␣
↪registerForActivityResult(multiplePermissionsContract, isGranted -> {
        updateDesign();
    });

    buttonRequest.setOnClickListener(v -> {
        multiplePermissionLauncher.launch(PERMISSIONS);
    });

    buttonRevoke.setOnClickListener(v -> {
        revokeSelfPermissionsOnKill(Arrays.asList(PERMISSIONS));
    });
```

In the above example, two permissions are revoked at the same time, but to revoke them individually, use revokeSelfPermissionOnKill as follows.

```
revokeSelfPermissionOnKill(Manifest.permission.CAMERA);
```

The sample program contains "@RequiresApi(api = 33)" because apps that use revokeSelfPermissionOnKill or revokeSelfPermissionsOnKill are not buildable with Compile SDK 32 or lower.

The timing of actual revocation of the permission is asynchronous, and revocation occurs when the system determines that it is safe to do so. If revoking immediately, safety can be ensured by prompting the user to restart the app and executing System.exit().

5.2.3.9 SMS OTP (One-Time Password) Protection

In recent years, security threats such as SIM swapping (an attack where an attacker hijacks someone else's phone number), phishing (tricking users into entering OTPs or passwords), SMS-forwarding malware, and abuse of accessibility permissions have led to increased incidents of unauthorized acquisition of SMS-delivered One-Time Passwords (OTPs). To address these security concerns, Android 17 introduces enhanced system-level protections that restrict access to SMS OTPs under system control to prevent credential theft by malicious applications.

Specifically, even if an application holds the READ_SMS or RECEIVE_SMS permissions, access to SMS messages containing OTPs is restricted and delayed for up to **3 hours** after receipt. During this 3-hour delay, the SMS_RECEIVED_ACTION broadcast is postponed, and queries to the SMS Provider database are filtered to exclude the target message. While default SMS applications are exempt from this restriction, applications that directly monitor the SMS inbox using the READ_SMS permission to programmatically and automatically retrieve OTPs will be significantly affected.

The scope of this protection is applied as follows:

- **All Applications (Regardless of target API level)**: The protection is applied to messages in the WebOTP format. Unless the application is verified as the intended recipient through domain verification, access to the message is delayed for 3 hours.
- **Applications Targeting Android 17 or Higher**: The protection is expanded to standard SMS messages (non-WebOTP and non-SMS Retriever formats), subjecting them to the same 3-hour delay.

Recommended Migration and Implementation

In alignment with the OS policy of restricting broad SMS reading to prevent unauthorized OTP interception, applications that require OTP extraction from SMS messages are strongly encouraged to migrate to official Google APIs that do not require broad SMS permissions, such as the permission-free "SMS Retriever API" or the user-consent-based "SMS User Consent API."

Table 5.2.2: Features and Comparison of Recommended APIs

Item	SMS Retriever API	SMS User Consent API
User Interaction	None (Fully automated)	Required (1-tap consent)
Permission Requirements	None	None
SMS Format	App Hash (11 characters) required at the end of the message	No restriction (Can retrieve any format)
Reliability and Control	High (If message format can be controlled internally)	High (Can retrieve SMS from third-party services)

If you can control the SMS delivery server and append the app's cryptographic hash to the message format, the SMS Retriever API is highly recommended to preserve a seamless, fully automated user experience. On the other hand, if you use third-party SMS delivery services where the message format cannot be controlled, you should implement the SMS User Consent API, which allows the application to retrieve a single SMS message after obtaining the user's explicit consent (a 1-tap dialog).

Verification Example of OS Behavior Changes

The 3-hour delay block introduced in Android 17 can be verified by testing a prototype application that attempts to read SMS messages using the traditional READ_SMS permission. The following code snippets demonstrate attempts to read SMS messages using both broadcast receiver (SMS_RECEIVED_ACTION) and content provider (Telephony.Sms.CONTENT_URI) queries.

When the app's targetSdkVersion is set to 36 (Android 16), the OTP can be read immediately upon SMS receipt. However, when the targetSdkVersion is updated to 37 (Android 17), the OTP cannot be read immediately and is blocked (delayed for 3 hours).

```
// 1. Verification code for receiving broadcast (SMS_RECEIVED_ACTION) (Excerpt)
override fun onReceive(context: Context, intent: Intent) {
    if (intent.action == Telephony.Sms.Intents.SMS_RECEIVED_ACTION) {
        val messages = Telephony.Sms.Intents.getMessagesFromIntent(intent)
        for (message in messages) {
            val body = message.messageBody
            // Process to extract OTP from the message body
            // *In Android 17 (Target SDK 37), this is not called until 3 hours␣
↪have elapsed after receipt
        }
    }
}
```

```
// 2. Verification code for reading from Content Provider (Excerpt)
val cursor = context.contentResolver.query(
    Telephony.Sms.CONTENT_URI,
    arrayOf(Telephony.Sms.BODY),
    null, null,
    Telephony.Sms.DATE + " DESC"
)
cursor?.use {
    if (it.moveToFirst()) {
        val body = it.getString(0)
        // Process to extract OTP from the message body
        // *In Android 17 (Target SDK 37), the corresponding message cannot be␣
↪retrieved until 3 hours have elapsed after receipt
    }
}
```

- **Caution Regarding Verification Code:** This code is strictly for verifying the OS behavior changes. In commercial production applications, you must migrate to the official APIs such as the SMS Retriever API as described above. Additionally, when conducting verification, ensure that no sensitive credentials (such as the SMS body or OTP) are output to the logs (e.g., Log.d), in strict compliance with the rules in 4.8.2.1. *Sensitive Information Must Not Be Included in Operation Log Information (Required)*.

5.3 Add In-house Accounts to Account Manager

Account Manager is the Android OS's system which centrally manages account information (account name, password) which is necessary for applications to access to online service and authentication token[10]. A user needs to register the account information to Account Manager in advance, and when an application tries to access to online service, Account Manager will automatically provide application authentication token after getting user's permission. The advantage of Account Manager is that an application doesn't need to handle the extremely sensitive information, password.

The structure of account management function which uses Account Manager is as per below Fig. 5.3.1. "Requesting application" is the application which accesses the online service, by getting authentication token, and this is above mentioned application. On the other hand, "Authenticator application" is function extension of Account Manager, and by providing Account Manager of an object called Authenticator, as a result Account Manager can manage centrally the account information and authentication token of the online service. Requesting application and Authenticator application don't need to be the separate ones, so these can be implemented as a single application.

[10] Account Manager provides mechanism of synchronizing with online services, however, this section doesn't deal with it.

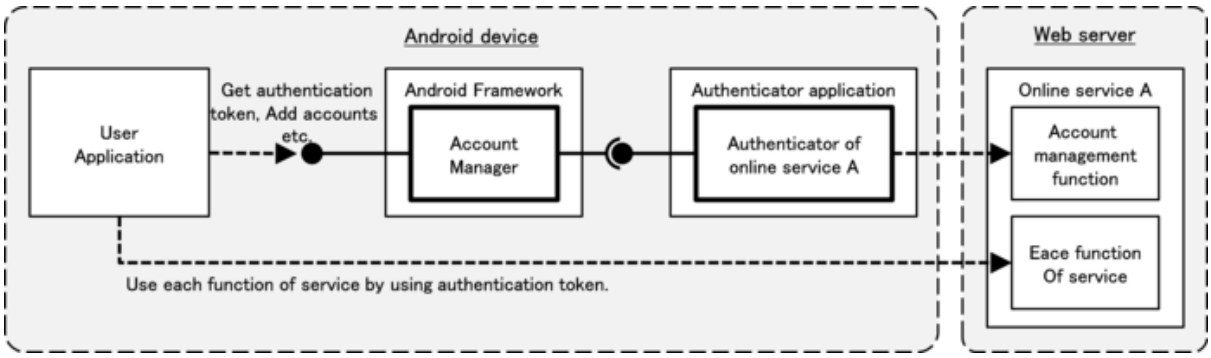


Fig. 5.3.1: Configuration of account management function which uses Account Manager

5.3.1 Sample Code

"5.3.1.1. *Creating In-house accounts*" is prepared as a sample of Authenticator application, and "5.3.1.2. *Using In-house Accounts*" is prepared as a sample of requesting application. In sample code set which is distributed in JSSEC's Web site, each of them is corresponded to AccountManager Authenticator and AccountManager User.

5.3.1.1 Creating In-house accounts

Here is the sample code of Authenticator application which enables Account Manager to use the in-house account. There is no Activity which can be launched from home screen in this application. Please pay attention that it's called indirectly via Account Manager from another sample code "5.3.1.2. *Using In-house Accounts*"

Points:

1. The service that provides an authenticator must be private.
2. The login screen activity must be implemented in an authenticator application.
3. The login screen activity must be made as a public activity.
4. The explicit intent which the class name of the login screen activity is specified must be set to KEY_INTENT.
5. Sensitive information (like account information or authentication token) must not be output to the log.
6. Password should not be saved in Account Manager.
7. HTTPS should be used for communication between an authenticator and the online services.

Service which gives Account Manager IBinder of Authenticator is defined in AndroidManifest.xml. Specify resource XML file which Authenticator is written, by meta-data.

```
AccountManager Authenticator/AndroidManifest.xml
<manifest xmlns:android="http://schemas.android.com/apk/res/android"

          xmlns:tools="http://schemas.android.com/tools">

  <!-- Necessary Permission to implement Authenticator -->
  <uses-permission android:name="android.permission.GET_ACCOUNTS" />
  <uses-permission android:name="android.permission.AUTHENTICATE_ACCOUNTS" />

  <application
      android:allowBackup="false"
      android:icon="@drawable/ic_launcher"
      android:label="@string/app_name" >

    <!-- Service which gives IBinder of Authenticator to AccountManager -->
```

(continues on next page)

(continued from previous page)

```
    <!-- *** POINT 1 *** The service that provides an authenticator must be␣
↪private. -->
    <service
        android:name=".AuthenticationService"
        android:exported="false" >
      <!-- intent-filter and meta-data are usual pattern. -->
      <intent-filter>
        <action android:name="android.accounts.AccountAuthenticator" />
      </intent-filter>
      <meta-data
          android:name="android.accounts.AccountAuthenticator"
          android:resource="@xml/authenticator" />
    </service>

    <!-- Activity for for login screen which is displayed when adding an account --
↪>
    <!-- *** POINT 2 *** The login screen activity must be implemented in an␣
↪authenticator application. -->
    <!-- *** POINT 3 *** The login screen activity must be made as a public␣
↪activity. -->
    <activity
        android:name=".LoginActivity"
        android:exported="true"
        android:label="@string/login_activity_title"
        android:theme="@android:style/Theme.Dialog"
        tools:ignore="ExportedActivity" />
  </application>

</manifest>
```

Define Authenticator by XML file. Specify account type etc. of in-house account.

```
res/xml/authenticator.xml
<account-authenticator xmlns:android="http://schemas.android.com/apk/res/android"
    android:accountType="org.jssec.android.accountmanager"
    android:icon="@drawable/ic_launcher"
    android:label="@string/label"
    android:smallIcon="@drawable/ic_launcher"
    android:customTokens="true" />
```

Service which gives Authenticator's Instance to AccountManager. Easy implementation which returns Instance of JssecAuthenticator class that is Authenticator implemented in this sample by onBind(), is enough.

```
AuthenticationService.java
/*
 * Copyright (C) 2012-2026 Japan Smartphone Security Association
 *
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 *
 *      http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
```

(continues on next page)

(continued from previous page)

```
 * limitations under the License.
 */

package org.jssec.android.accountmanager.authenticator;

import android.app.Service;
import android.content.Intent;
import android.os.IBinder;

public class AuthenticationService extends Service {

    private JssecAuthenticator mAuthenticator;

    @Override
    public void onCreate() {
        mAuthenticator = new JssecAuthenticator(this);
    }

    @Override
    public IBinder onBind(Intent intent) {
        return mAuthenticator.getIBinder();
    }
}
```

JssecAuthenticator is the Authenticator which is implemented in this sample. It inherits AbstractAccountAuthenticator, and all abstract methods are implemented. These methods are called by Account Manager. At addAccount() and at getAuthToken(), the intent for launching LoginActivity to get authentication token from online service are returned to Account Manager.

JssecAuthenticator.java

```
/*
 * Copyright (C) 2012-2026 Japan Smartphone Security Association
 *
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 *
 *      http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */

package org.jssec.android.accountmanager.authenticator;

import android.accounts.AbstractAccountAuthenticator;
import android.accounts.Account;
import android.accounts.AccountAuthenticatorResponse;
import android.accounts.AccountManager;
import android.accounts.NetworkErrorException;
import android.content.Context;
import android.content.Intent;
import android.os.Bundle;
```

(continues on next page)

(continued from previous page)

```
public class JssecAuthenticator extends AbstractAccountAuthenticator {

    public static final String JSSEC_ACCOUNT_TYPE =
            "org.jssec.android.accountmanager";
    public static final String JSSEC_AUTHTOKEN_TYPE = "webservice";
    public static final String JSSEC_AUTHTOKEN_LABEL = "JSSEC Web Service";
    public static final String RE_AUTH_NAME = "reauth_name";

    protected final Context mContext;

    public JssecAuthenticator(Context context) {
        super(context);
        mContext = context;
    }

    @Override
    public Bundle addAccount(AccountAuthenticatorResponse response,
                             String accountType, String authTokenType,
                             String[] requiredFeatures, Bundle options)
            throws NetworkErrorException {

        AccountManager am = AccountManager.get(mContext);
        Account[] accounts = am.getAccountsByType(JSSEC_ACCOUNT_TYPE);
        Bundle bundle = new Bundle();
        if (accounts.length > 0) {
            // In this sample code, when an account already exists, consider it
            // as an error.
            bundle.putString(AccountManager.KEY_ERROR_CODE, String.valueOf(-1));
            bundle.putString(AccountManager.KEY_ERROR_MESSAGE,
                             mContext.getString(R.string.error_account_exists));
        } else {
            // *** POINT 2 *** The login screen activity must be implemented in an
            // authenticator application.
            // *** POINT 4 *** The explicit intent which the class name of the
            // login screen activity is specified must be set to KEY_INTENT.
            Intent intent = new Intent(mContext, LoginActivity.class);
            intent.putExtra(AccountManager.KEY_ACCOUNT_AUTHENTICATOR_RESPONSE,␣
↪response);

            bundle.putParcelable(AccountManager.KEY_INTENT, intent);
        }
        return bundle;
    }

    @Override
    public Bundle getAuthToken(AccountAuthenticatorResponse response,
                               Account account, String authTokenType,
                               Bundle options)
            throws NetworkErrorException {

        Bundle bundle = new Bundle();
        if (accountExist(account)) {
            // *** POINT 4 *** The explicit intent which the class name of the
            // login screen activity is specified must be set to KEY_INTENT.
            Intent intent = new Intent(mContext, LoginActivity.class);
```

(continues on next page)

(continued from previous page)

```
            intent.putExtra(RE_AUTH_NAME, account.name);
            intent.putExtra(AccountManager.KEY_ACCOUNT_AUTHENTICATOR_RESPONSE,
                            response);
            bundle.putParcelable(AccountManager.KEY_INTENT, intent);
        } else {
            // When the specified account doesn't exist, consider it as an error.
            bundle.putString(AccountManager.KEY_ERROR_CODE, String.valueOf(-2));
            bundle.putString(AccountManager.KEY_ERROR_MESSAGE,
                    mContext.getString(R.string.error_account_not_exists));
        }
        return bundle;
    }

    @Override
    public String getAuthTokenLabel(String authTokenType) {
        return JSSEC_AUTHTOKEN_LABEL;
    }

    @Override
    public Bundle confirmCredentials(AccountAuthenticatorResponse response,
                                     Account account, Bundle options)
            throws NetworkErrorException {
        return null;
    }

    @Override
    public Bundle editProperties(AccountAuthenticatorResponse response,
                                 String accountType) {
        return null;
    }

    @Override
    public Bundle updateCredentials(AccountAuthenticatorResponse response,
                                    Account account,
                                    String authTokenType, Bundle options)
            throws NetworkErrorException {
        return null;
    }

    @Override
    public Bundle hasFeatures(AccountAuthenticatorResponse response,
                              Account account, String[] features)
            throws NetworkErrorException {
        Bundle result = new Bundle();
        result.putBoolean(AccountManager.KEY_BOOLEAN_RESULT, false);
        return result;
    }

    private boolean accountExist(Account account) {
        AccountManager am = AccountManager.get(mContext);
        Account[] accounts = am.getAccountsByType(JSSEC_ACCOUNT_TYPE);
        for (Account ac : accounts) {
            if (ac.equals(account)) {
                return true;
            }
        }
```

(continues on next page)

(continued from previous page)

```
        return false;
    }
}
```

This is Login activity which sends an account name and password to online service, and perform login authentication, and as a result, get an authentication token. It's displayed when adding a new account or when getting authentication token again. It's supposed that the actual access to online service is implemented in WebService class.

LoginActivity.java

```
/*
 * Copyright (C) 2012-2026 Japan Smartphone Security Association
 *
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 *
 *      http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */

package org.jssec.android.accountmanager.authenticator;

import org.jssec.android.accountmanager.webservice.WebService;

import android.accounts.Account;
import android.accounts.AccountAuthenticatorActivity;
import android.accounts.AccountManager;
import android.content.Intent;
import android.os.Bundle;
import android.text.InputType;
import android.text.TextUtils;
import android.util.Log;
import android.view.View;
import android.view.Window;
import android.widget.EditText;

public class LoginActivity extends AccountAuthenticatorActivity {
    private static final String TAG =
            AccountAuthenticatorActivity.class.getSimpleName();
    private String mReAuthName = null;
    private EditText mNameEdit = null;
    private EditText mPassEdit = null;

    @Override
    public void onCreate(Bundle icicle) {
        super.onCreate(icicle);

        // Display alert icon
        requestWindowFeature(Window.FEATURE_LEFT_ICON);
        setContentView(R.layout.login_activity);
        getWindow().setFeatureDrawableResource(Window.FEATURE_LEFT_ICON,
```

(continues on next page)

(continued from previous page)

```
                                                android.R.drawable.ic_dialog_alert);

        // Find a widget in advance
        mNameEdit = (EditText) findViewById(R.id.username_edit);
        mPassEdit = (EditText) findViewById(R.id.password_edit);

        // *** POINT 3 *** The login screen activity must be made as a public
        // activity, and suppose the attack access from other application.
        // Regarding external input, only RE_AUTH_NAME which is String type of
        // Intent#extras, are handled.
        // This external input String is passed toextEdit#setText(),
        // WebService#login(),new Account(), as a parameter,it's verified that
        // there's no problem if any character string is passed.
        mReAuthName = getIntent().getStringExtra(JssecAuthenticator.RE_AUTH_NAME);
        if (mReAuthName != null) {
            // Since LoginActivity is called with the specified user name,
            // user name should not be editable.
            mNameEdit.setText(mReAuthName);
            mNameEdit.setInputType(InputType.TYPE_NULL);
            mNameEdit.setFocusable(false);
            mNameEdit.setEnabled(false);
        }
    }

    // It's executed when login button is pressed.
    public void handleLogin(View view) {
        String name = mNameEdit.getText().toString();
        String pass = mPassEdit.getText().toString();

        if (TextUtils.isEmpty(name) || TextUtils.isEmpty(pass)) {
            // Process when the inputed value is incorrect
            setResult(RESULT_CANCELED);
            finish();
        }

        // Login to online service based on the inpputted account information.
        WebService web = new WebService();
        String authToken = web.login(name, pass);
        if (TextUtils.isEmpty(authToken)) {
            // Process when authentication failed
            setResult(RESULT_CANCELED);
            finish();
        }

        // Process when login was successful, is as per below.

        // *** POINT 5 *** Sensitive information (like account information or
        // authentication token) must not be output to the log.
        Log.i(TAG, "WebService login succeeded");


        if (mReAuthName == null) {
            // Register accounts which logged in successfully, to aAccountManager
            // *** POINT 6 *** Password should not be saved in Account Manager.
            AccountManager am = AccountManager.get(this);
            Account account =
```

(continues on next page)

(continued from previous page)

```
                    new Account(name, JssecAuthenticator.JSSEC_ACCOUNT_TYPE);
            am.addAccountExplicitly(account, null, null);
            am.setAuthToken(account, JssecAuthenticator.JSSEC_AUTHTOKEN_TYPE,
                            authToken);
            Intent intent = new Intent();
            intent.putExtra(AccountManager.KEY_ACCOUNT_NAME, name);
            intent.putExtra(AccountManager.KEY_ACCOUNT_TYPE,
                            JssecAuthenticator.JSSEC_ACCOUNT_TYPE);
            setAccountAuthenticatorResult(intent.getExtras());
            setResult(RESULT_OK, intent);
        } else {
            // Return authentication token
            Bundle bundle = new Bundle();
            bundle.putString(AccountManager.KEY_ACCOUNT_NAME, name);
            bundle.putString(AccountManager.KEY_ACCOUNT_TYPE,
                             JssecAuthenticator.JSSEC_ACCOUNT_TYPE);
            bundle.putString(AccountManager.KEY_AUTHTOKEN, authToken);
            setAccountAuthenticatorResult(bundle);
            setResult(RESULT_OK);
        }
        finish();
    }
}
```

Actually, WebService class is dummy implementation here, and this is the sample implementation which supposes authentication is always successful, and fixed character string is returned as an authentication token.

WebService.java

```
/*
 * Copyright (C) 2012-2026 Japan Smartphone Security Association
 *
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 *
 *      http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */

package org.jssec.android.accountmanager.webservice;

public class WebService {

    /**
     * Suppose to access to account managemnet function of online service.
     *
     * @param username Account name character string
     * @param password password character string
     * @return Return authentication token
     */
    public String login(String username, String password) {
```

(continues on next page)

(continued from previous page)

```
        // *** POINT 7 *** HTTPS should be used for communication between an
        // authenticator and the online services.
        // Actually, communication process with servers is implemented here,
        // but Omit here, since this is a sample.
        return getAuthToken(username, password);
    }

    private String getAuthToken(String username, String password) {
        // In fact, get the value which uniqueness and impossibility of
        // speculation are guaranteed by the server, but the fixed value
        // is returned without communication here, since this is sample.
        return "c2f981bda5f34f90c0419e171f60f45c";
    }
}
```

5.3.1.2 Using In-house Accounts

Here is the sample code of an application which adds an in-house account and gets an authentication token. When another sample application "5.3.1.1. *Creating In-house accounts*" is installed in a device, in-house account can be added or authentication token can be got. "Access request" screen is displayed only when the signature keys of both applications are different.

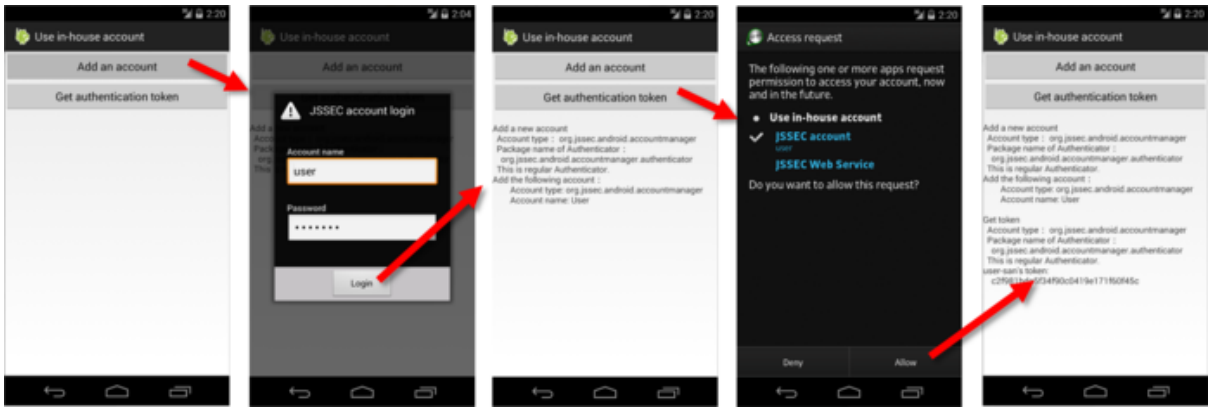


Fig. 5.3.2: Behavior screen of sample application AccountManager User

Point:

1. Execute the account process after verifying if the authenticator is regular one.

AndroidManifest.xml of AccountManager user application. Declare to use necessary Permission. Refer to "5.3.3.1. *Usage of Account Manager and Permission*" for the necessary Permission.

```
AccountManager User/AndroidManifest.xml
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
          >

  <uses-permission android:name="android.permission.GET_ACCOUNTS" />
  <uses-permission android:name="android.permission.MANAGE_ACCOUNTS" />
  <uses-permission android:name="android.permission.USE_CREDENTIALS" />

  <queries>
    <package android:name="org.jssec.android.accountmanager.authenticator" />
  </queries>
```

(continues on next page)

(continued from previous page)

```
  <application
      android:allowBackup="false"
      android:icon="@drawable/ic_launcher"
      android:label="@string/app_name"
      android:theme="@style/AppTheme" >
    <activity
        android:name=".UserActivity"
        android:label="@string/app_name"
        android:exported="true" >
      <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER" />
      </intent-filter>
    </activity>
  </application>

</manifest>
```

Activity of user application. When tapping the button on the screen, either addAccount() or getAuthToken() is to be executed. Authenticator which corresponds to the specific account type may be fake in some cases, so pay attention that the account process is started after verifying that the Authenticator is regular one.

UserActivity.java

```
/*
 * Copyright (C) 2012-2026 Japan Smartphone Security Association
 *
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 *
 *      http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */

package org.jssec.android.accountmanager.user;

import java.io.IOException;

import org.jssec.android.shared.PkgCert;
import org.jssec.android.shared.Utils;

import android.accounts.Account;
import android.accounts.AccountManager;
import android.accounts.AccountManagerCallback;
import android.accounts.AccountManagerFuture;
import android.accounts.AuthenticatorDescription;
import android.accounts.AuthenticatorException;
import android.accounts.OperationCanceledException;
import android.app.Activity;
import android.content.Context;
```

(continues on next page)

(continued from previous page)

```
import android.os.Bundle;
import android.view.View;
import android.widget.TextView;

public class UserActivity extends Activity {

    // Information of the Authenticator to be used
    private static final String JSSEC_ACCOUNT_TYPE =
            "org.jssec.android.accountmanager";
    private static final String JSSEC_TOKEN_TYPE = "webservice";
    private TextView mLogView;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.user_activity);

        mLogView = (TextView)findViewById(R.id.logview);
    }

    public void addAccount(View view) {
        logLine();
        logLine("Add a new account");

        // *** POINT 1 *** Execute the account process after verifying if the
        // authenticator is regular one.
        if (!checkAuthenticator()) return;

        AccountManager am = AccountManager.get(this);
        am.addAccount(JSSEC_ACCOUNT_TYPE, JSSEC_TOKEN_TYPE, null, null, this,
            new AccountManagerCallback<Bundle>() {
                @Override
                public void run(AccountManagerFuture<Bundle> future) {
                    try {
                        Bundle result = future.getResult();
                        String type =
                            result.getString(AccountManager.KEY_ACCOUNT_TYPE);
                        String name =
                            result.getString(AccountManager.KEY_ACCOUNT_NAME);
                        if (type != null && name != null) {
                            logLine("Add the following accounts:");
                            logLine("  Account type: %s", type);
                            logLine("  Account name: %s", name);
                        } else {
                            String code =
                                result.getString(AccountManager.KEY_ERROR_CODE);
                            String msg =
                                result.getString(AccountManager.KEY_ERROR_MESSAGE);
                            logLine("The account cannot be added");
                            logLine("  Error code %s: %s", code, msg);
                        }
                    } catch (OperationCanceledException e) {
                    } catch (AuthenticatorException e) {
                    } catch (IOException e) {
                    }
                }
```

(continues on next page)

(continued from previous page)

```
            },
        null);
    }

    public void getAuthToken(View view) {
        logLine();
        logLine("Get token");

        // *** POINT 1 *** After checking that the Authenticator is the regular
        // one, execute account process.
        if (!checkAuthenticator()) return;

        AccountManager am = AccountManager.get(this);
        Account[] accounts = am.getAccountsByType(JSSEC_ACCOUNT_TYPE);
        if (accounts.length > 0) {
            Account account = accounts[0];
            am.getAuthToken(account, JSSEC_TOKEN_TYPE, null, this,
                new AccountManagerCallback<Bundle>() {
                    @Override
                    public void run(AccountManagerFuture<Bundle> future) {
                        try {
                            Bundle result = future.getResult();
                            String name =
                                result.getString(AccountManager.KEY_ACCOUNT_NAME);
                            String authtoken =
                                result.getString(AccountManager.KEY_AUTHTOKEN);
                            logLine("%s-san's token:", name);
                            if (authtoken != null) {
                                logLine("    %s", authtoken);
                            } else {
                                logLine("    Couldn't get");
                            }
                        } catch (OperationCanceledException e) {
                            logLine("  Exception: %s",e.getClass().getName());
                        } catch (AuthenticatorException e) {
                            logLine("  Exception: %s",e.getClass().getName());
                        } catch (IOException e) {
                            logLine("  Exception: %s",e.getClass().getName());
                        }
                    }
                },
            null);
        } else {
            logLine("Account is not registered.");
        }
    }

    // *** POINT 1 *** Verify that Authenticator is regular one.
    private boolean checkAuthenticator() {
        AccountManager am = AccountManager.get(this);
        String pkgname = null;
        for (AuthenticatorDescription ad : am.getAuthenticatorTypes()) {
            if (JSSEC_ACCOUNT_TYPE.equals(ad.type)) {
                pkgname = ad.packageName;
                break;
            }
```

(continues on next page)

(continued from previous page)

```
        }

        if (pkgname == null) {
            logLine("Authenticator cannot be found.");
            return false;
        }

        logLine("  Account type: %s", JSSEC_ACCOUNT_TYPE);
        logLine("  Package name of Authenticator: ");
        logLine("    %s", pkgname);

        if (!PkgCert.test(this, pkgname, getTrustedCertificateHash(this))) {
            logLine("  It's not regular Authenticator(certificate is not matched.)
↪");
            return false;
        }

        logLine("  This is regular Authenticator.");
        return true;
    }

    // Certificate hash value of regular Authenticator application
    // Certificate hash value can be checked in sample applciation
    // JSSEC CertHash Checker
    private String getTrustedCertificateHash(Context context) {
        if (Utils.isDebuggable(context)) {
            // Certificate hash value of  debug.keystore "androiddebugkey"
            return "0EFB7236 328348A9 89718BAD DF57F544 D5CCB4AE B9DB34BC 1E29DD26␣
↪F77C8255";
        } else {
            // Certificate hash value of  keystore "my company key"
            return "D397D343 A5CBC10F 4EDDEB7C A10062DE 5690984F 1FB9E88B D7B3A7C2␣
↪42E142CA";
        }
    }

    private void log(String str) {
        mLogView.append(str);
    }

    private void logLine(String line) {
        log(line + "\n");
    }

    private void logLine(String fmt, Object... args) {
        logLine(String.format(fmt, args));
    }

    private void logLine() {
        log("\n");
    }
}
```

PkgCert.java

```
/*
 * Copyright (C) 2012-2026 Japan Smartphone Security Association
```

(continues on next page)

(continued from previous page)

```
 *
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 *
 *      http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */

package org.jssec.android.shared;

import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;

import android.content.Context;
import android.content.pm.PackageInfo;
import android.content.pm.PackageManager;
import android.content.pm.PackageManager.NameNotFoundException;
import android.content.pm.Signature;

public class PkgCert {

    public static boolean test(Context ctx, String pkgname, String correctHash) {
        if (correctHash == null) return false;
        correctHash = correctHash.replaceAll(" ", "");
        return correctHash.equals(hash(ctx, pkgname));
    }

    public static String hash(Context ctx, String pkgname) {
        if (pkgname == null) return null;
        try {
            PackageManager pm = ctx.getPackageManager();
            PackageInfo pkginfo =
                pm.getPackageInfo(pkgname, PackageManager.GET_SIGNATURES);
            // Will not handle multiple signatures.
            if (pkginfo.signatures.length != 1) return null;
            Signature sig = pkginfo.signatures[0];
            byte[] cert = sig.toByteArray();
            byte[] sha256 = computeSha256(cert);
            return byte2hex(sha256);
        } catch (NameNotFoundException e) {
            return null;
        }
    }

    private static byte[] computeSha256(byte[] data) {
        try {
            return MessageDigest.getInstance("SHA-256").digest(data);
        } catch (NoSuchAlgorithmException e) {
            return null;
        }
```

(continues on next page)

(continued from previous page)

```
    }

    private static String byte2hex(byte[] data) {
        if (data == null) return null;
        final StringBuilder hexadecimal = new StringBuilder();
        for (final byte b : data) {
            hexadecimal.append(String.format("%02X", b));
        }
        return hexadecimal.toString();
    }
}
```

5.3.2 Rule Book

Follow the rules below when implementing Authenticator application.

1. *Service that Provides Authenticator Must Be Private (Required)*
2. *Login Screen Activity Must Be Implemented by Authenticator Application (Required)*
3. *The Login Screen Activity Must Be Made as a Public Activity and Suppose Attack Accesses by Other Applications (Required)*
4. *Provide KEY_INTENT with Explicit Intent with the Specified Class Name of Login Screen Activity (Required)*
5. *Sensitive Information (like Account Information and Authentication Token) Must Not Be Output to the Log (Required)*
6. *Password Should Not Be Saved in Account Manager (Recommended)*
7. *HTTPS Should Be Used for Communication Between an Authenticator and the Online Service (Required)*

Follow the rules below when implementing user application.

8. *Account Process Should Be Executed after verifying if the Authenticator is the regular one (Required)*

5.3.2.1 Service that Provides Authenticator Must Be Private (Required)

It's presupposed that the Service which provides with Authenticator is used by Account Manager, and it should not be accessed by other applications. So, by making it Private Service, it can exclude accesses by other applications. In addition, Account Manager runs with system privilege, so Account Manager can access even if it's private Service.

5.3.2.2 Login Screen Activity Must Be Implemented by Authenticator Application (Required)

Login screen for adding a new account and getting the authentication token should be implemented by Authenticator application. Own Login screen should not be prepared in user application side. As mentioned at the beginning of this article, "The advantage of AccountManager is that the extremely sensitive information/password is not necessarily to be handled by application.", If login screen is prepared in user application side, password is handled by user application, and its design becomes what is beyond the policy of Account Manager.

By preparing login screen by Authenticator application, who can operate login screen is limited only the device's user. It means that there's no way to attack the account for malicious applications by attempting to login directly, or by creating an account.

5.3.2.3 The Login Screen Activity Must Be Made as a Public Activity and Suppose Attack Accesses by Other Applications (Required)

Login screen Activity is the system launched by the user application's p. In order that the login screen Activity is displayed even when the signature keys of user application and Authenticator application are different, login screen Activity should be implemented as Public Activity.

What login screen Activity is public Activity means, that there's a chance that it may be launched by malicious applications. Never trust on any input data. Hence, it's necessary to take the counter-measures mentioned in "3.2. *Handling Input Data Carefully and Securely*"

5.3.2.4 Provide KEY_INTENT with Explicit Intent with the Specified Class Name of Login Screen Activity (Required)

When Authenticator needs to open login screen Activity, Intent which launches login screen Activity is to be given in the Bundle that is returned to Account Manager, by KEY_INTENT. The Intent to be given, should be the explicit Intent which specifies class name of login screen Activity. If an *implicit* Intent is given, the framework may attempt to launch an Activity *other* than the Activity prepared by the Authenticator app for the login window. On Android 4.4 (API Level 19) and later versions, this may cause the app to crash; on earlier versions it may cause unintended Activities prepared by other apps to be launched.

On Android 4.4(API Level 19) and later versions, if the signature of an app launched by an intent given by the framework via KEY_INTENT does not match the signature of the Authenticator app, a SecurityException is generated; in this case, there is no risk that a false login screen will be launched; however, there is a possibility that the ordinary screen will be able to launch and the user's normal use of the app will be obstructed. On versions prior to Android 4.4(API Level 19), there is a risk that a false login screen prepared by a malicious app will be launched, and thus that the user may input passwords and other authentication information to the malicious app.

5.3.2.5 Sensitive Information (like Account Information and Authentication Token) Must Not Be Output to the Log (Required)

Applications which access to online service sometimes face a trouble like it cannot access to online service successfully. The causes of unsuccessful access are various, like lack in network environment arrangement, mistakes in implementing communication protocol, lack of Permission, authentication error, etc. A common implementation is that a program outputs the detailed information to log, so that developer can analyze the cause of a problem later.

Sensitive information like password or authentication token should not be output to log. Log information can be read from other applications, so it may become the cause of information leakage. Also, account names should not be output to log, if it could be lead the damage of leakage.

5.3.2.6 Password Should Not Be Saved in Account Manager (Recommended)

Two of authentication information, password and authentication token, can be saved in an account to be register to AccountManager. This information is to be stored in accounts.db under the following directories, in a plain text (i.e. without encryption).

- Android 4.1 or earlier
 /data/system/accounts.db
- Android 4.2 to Android 6.0
 /data/system/0/accounts.db or /data/system/<UserId>/accounts.db
- Android 7.0 or later
 /data/system_ce/0/accounts_ce.db

Note: Because multiuser functionality is supported on Android 4.2 and later versions, this has been changed to save the content to a user-specific directory. Also, because Android 7.0 and later versions support Direct Boot, the database file is divided into two parts: one file that handles data while locked (/data/system_de/0/accounts_de_db) and a separate file that handles data while unlocked (/data/system_ce/0/accounts_ce.db) Under ordinary circumstances, authentication information is stored in the latter database file.

Root privileges or system privileges are required to read the content of these database files, so they cannot be read on commercial Android terminals. If Android OS contains any vulnerabilities that allow attackers to acquire root privileges or system privileges, this would leave the authentication information stored in accounts.db exposed to risk.

To read in the contents of accounts.db, either root privilege or system privilege is required, and it cannot be read from the marketed Android devices. In the case there is any vulnerability in Android OS, which root privilege or system privilege may be taken over by attackers, authentication information which is saved in accounts.db will be on the edge of the risk.

The Authentication application, which is introduced in this article, is designed to save authentication token in AccountManager without saving user password. When accessing to online service continuously in a certain period, generally the expiration period of authentication token is extended, so the design that password is not saved is enough in most cases.

In general, valid date of authentication token is shorter than password, and it's characteristic that it can be disabled anytime. In case, authentication token is leaked, it can be disabled, so authentication token is comparatively safer, compared with password. In the case authentication token is disabled, user can input the password again to get a new authentication token.

If disabling password when it's leaked, user cannot use online service any more. In this case, it requires call center support etc., and it will take huge cost. Hence, it's better to avoid from the design to save password in AccountManager. In case, the design to save password cannot be avoided, high level of reverse engineering counter-measures like encrypting password and obfuscating the key of that encryption, should be taken.

5.3.2.7 HTTPS Should Be Used for Communication Between an Authenticator and the Online Service (Required)

Password or authentication token is so called authentication information, and if it's taken over by the third party, the third party can masquerade as the valid user. Since Authenticator sends/receives these types of authentication information with online service, reliable encrypted communication method like an HTTPS should be used.

5.3.2.8 Account Process Should Be Executed after verifying if the Authenticator is the regular one (Required)

In the case there are several Authenticators which the same account type is defined in a device, Authenticator which was installed earlier becomes valid. So, when the own Authenticator was installed later, it's not to be used.

If the Authenticator which was installed earlier, is the malware's masquerade, account information inputted by user may be taken over by malware. User application should verify the account type which performs account operation, whether the regular Authenticator is allocated to it or not, before executing account operation.

Whether the Authenticator which is allocated to one account type is regular one or not, can be verified by checking whether the certificate hash value of the package of Authenticator matches with pre-confirmed valid certificate hash value. If the certificate hash values are found to be not matched, a measure to prompt user to uninstall the package which includes the unexpected Authenticator allocated to that account type, is preferable.

5.3.3 Advanced Topics

5.3.3.1 Usage of Account Manager and Permission

To use each method of AccountManager class, it's necessary to declare to use the appropriate Permission respectively, in application's AndroidManifest.xml. In Android 5.1 (API Level 22) and earlier versions, privileges such as AUTHENTICATE_ACCOUNTS, GET_ACCOUNTS, or MANAGE_ACCOUNTS are required; the privileges corresponding to various methods are shown in Table 5.3.1.

Table 5.3.1: Function of Account Manager and Permission

Functions that Account Manager provides		
Permission	**Method**	**Explanation**
AUTHENTICATE_ACCOUNTS (Only Packages which are Authenticator, can use.)	getPassword()	To get password
	getUserData()	To get user information
	addAccountExplicitly()	To add accounts to DB
	peekAuthToken()	To get cached token
	setAuthToken()	To register authentication token
	setPassword()	To change password
	setUserData()	To set user information
	renameAccount()	To rename account
GET_ACCOUNTS	getAccounts()	To get a list of all accounts
	getAccountsByType()	To get a list of all accounts which account types are same
	getAccountsByTypeAndFeatures()	To get a list of all accounts which have the specified function
	addOnAccountsUpdatedListener()	To register event listener
	hasFeatures()	Whether it has the specified function or not
MANAGE_ACCOUNTS	getAuthTokenByFeatures()	To get authentication token of the accounts which have the specified function
	addAccount()	To request a user to add accounts
	removeAccount()	To remove an account
	clearPassword()	Initialize password
	updateCredentials()	Request a user to change password
	editProperties()	Change Authenticator setting
	confirmCredentials()	Request a user to input password again
USE_CREDENTIALS	getAuthToken()	To get authentication token
	blockingGetAuthToken()	To get authentication token
MANAGE_ACCOUNTS or USE_CREDENTIALS	invalidateAuthToken()	To delete cached token

In case using methods group which AUTHENTICATE_ACCOUNTS Permission is necessary, there is a restriction related to package signature key along with Permission. Specifically, the key for signature of package that provides Authenticator and the key for signature of package in the application that uses methods, should be the same. So, when distributing an application which uses method group which AUTHENTICATE_ACCOUNTS Permission is necessary other than Authenticator, signature should be signed by the key which is the same as Authenticator.

In Android 6.0 (API Level 23) and later versions, Permissions other than GET_ACCOUNTS are not used, and there is no difference between what may be done whether or not it is declared. For methods that request AUTHENTICATE_ACCOUNTS on Android 5.1 (API Level 22) and earlier versions, note that—even if you wish to request a Permission—the call can only be made if signatures match (if the signatures do not match then a SecurityException is generated).

In addition, access controls for API routines that require GET_ACCOUNTS changed in Android 8.0 (API Level 26). In this and later versions, if the targetSdkVersion of the app on the side using the account information is 26 or higher, account information can generally not be obtained if the signature does not match that of the Authenticator app, even if GET_ACCOUNTS has been granted. However, if the Authenticator app calls the setAccountVisibility method to specify a package name, account information can be provided even to apps with non-matching signatures.

In a development phase by Android Studio, since a fixed debug keystore might be shared by some Android Studio projects, developers might implement and test Account Manager by considering only permissions and no signature. It's necessary for especially developers who use the different signature keys per applications, to be very careful when selecting which key to use for applications, considering this restriction. In addition, since the data which is obtained by AccountManager includes the sensitive information, so need to handle with care in order to decrease the risk like leakage or unauthorized use.

5.3.3.2 Cases in which Authenticator accounts with non-matching signatures may be read in Android 8.0 (API Level 26) or later

In Android 8.0 (API Level 26) and later versions, account-information-fetching methods that required GET_ACCOUNTS Permission in Android 7.1 (API Level 25) and earlier versions may now be called without that permission. Instead, account information may now be obtained only in cases where the signature matches or in which the setAccountVisibility method has been used on the Authenticator app side to specify an app to which account information may be provided However, note carefully that there are a number of exceptions to this rule, implemented by the framework. In what follows we discuss these exceptions.

First, when the targetSdkVersion of the app using the account information is 25 (Android 7.1) or below, the above rule does not apply; in this case apps with the GET_ACCOUNTS permission may obtain account information within the terminal regardless of its signature. However, below we discuss how this behavior may be changed depending on the Authenticator-side implementation. Next, account information for Authenticators that declare the use of WRITE_CONTACTS Permission may be read by other apps with READ_CONTACTS Permission, regardless of signature. This is not a bug, but is rather the way the framework is designed[11]. Note again that this behavior may differ depending on the Authenticator-side implementation.

Thus we see that there are some exceptional cases in which account information may be read even for apps with non-matching signatures and for which the setAccountVisibility method has not been called to specify a destination to which account information is to be provided. However, these behaviors may be modified by calling the setAccountVisibility method on the Authenticator side, as in the following snippet.

Do not provide account information to third-party apps

```
// account for which to change visibility
accountManager.setAccountVisibility(account,
    AccountManager.PACKAGE_NAME_KEY_LEGACY_VISIBLE,
    AccountManager.VISIBILITY_USER_MANAGED_NOT_VISIBLE);
```

By proceeding this way, we can avoid the framework's default behavior regarding account information for Authenticators that have called the setAccountVisibility method; the above modification ensures that account information is not provided even in cases where targetSdkVersion <= 25 or READ_CONTACTS permission is present.

5.4 Communicating via HTTPS

Most of smartphone applications communicate with Web servers on the Internet. As methods of communications, here we focus on the 2 methods of HTTP and HTTPS. From the security point of view, HTTPS communication is preferable. Lately, major Web services like Google or Facebook have been coming to use HTTPS as default. However, among HTTPS connection methods, those that use SSL3.0 / early Transport Layer Security (TLS) protocols are known to be susceptible to a vulnerability (commonly known as POODLE and BEAST), and we strongly recommend against the use of such methods, please refer to "5.4.3.8. *(Column): Transitioning to TLS1.2/TLS1.3 for secure connections*".

Since 2012, many defects in implementation of HTTPS communication have been pointed out in Android applications. These defects might have been implemented for accessing testing Web servers operated by server certificates that are not issued by trusted third party certificate authorities, but issued privately (hereinafter, called private certificates).

In this section, communication methods of HTTP and HTTPS are explained and the method to access safely with HTTPS to a Web server operated by a private certificate is also described.

[11] It is assumed that Authenticators that declare the use of WRITE_CONTACTS Permission will write account information to ContactsProvider, and that apps with READ_CONTACTS Permission will be granted permission to obtain account information.

5.4.1 Sample Code

You can find out which type of HTTP/HTTPS communication you are supposed to implement through the following chart (Fig. 5.4.1) shown below.

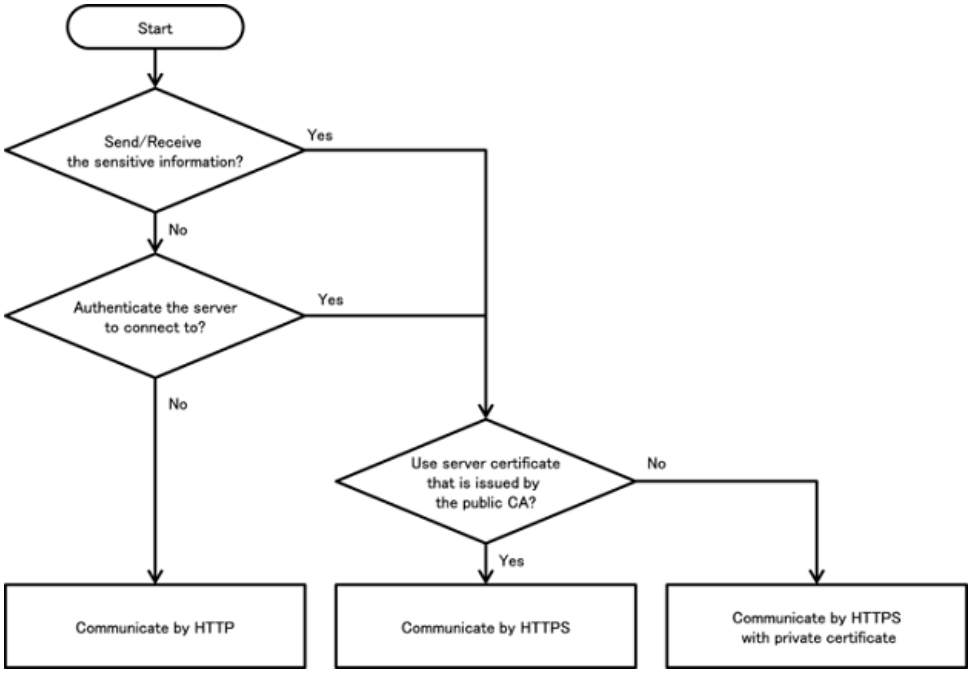


Fig. 5.4.1: Flow Figure to select sample code of HTTP/HTTPS

When sensitive information is sent or received, HTTPS communication is to be used because its communication channel is encrypted with SSL/TLS. HTTPS communication is required for the following sensitive information.

- Login ID/Password for Web services.
- Information for keeping authentication state (session ID, token, Cookie etc.)
- Important/confidential information depending on Web services (personal information, credit card information etc.)

A smartphone application with network communication is a part of "system" as well as a Web server. And you have to select HTTP or HTTPS for each communication based on secure design and coding considering the whole "system". Table 5.4.1 is for a comparison between HTTP and HTTPS. And Table 5.4.2 is for the differences in sample codes.

Table 5.4.1: Comparison between HTTP communication method and HTTPS communication method

		HTTP	HTTPS
Character-istics	URL	Starting with http://	Starting with https://
	Encrypting contents	Not available	Available
	Tampering detection of contents	Impossible	Possible
	Authenticating a server	Impossible	Possible
Damage Risk	Reading contents by attackers	High	Low
	Modifying contents by attackers	High	Low
	Application's access to a fake server	High	Low

Table 5.4.2: Explanation of HTTP/HTTPS communication Sample code

Sample code	Communication	Sending/Receiving sensitive information	Server certificate
Communicating via HTTP	HTTP	Not applicable	-
Communicating via HTTP	HTTPS	OK	Server certificates issued by trusted third party's certificate authorities like Cybertrust and VeriSign etc.
Communicating via HTTPS with private certificate	HTTPS	OK	Private certificate Operation mode which can be often seen in intra server or in test server.

Android supports java.net.HttpURLConnection/javax.net.ssl.HttpsURLConnection as HTTP/HTTPS communication APIs. Support for the Apache HttpClient, which is another HTTP client library, is removed at the release of the Android 6.0(API Level 23).

5.4.1.1 Communicating via HTTP

It must be assumed that all information sent and received via HTTP communication can potentially be eavesdropped on or tampered with by an attacker. Furthermore, the connected server may also be a fake server prepared by an attacker. HTTP communication can only be used for purposes where damage will not occur or remains within an acceptable range even under these assumptions.

The following sample code is an app that retrieves and displays images on a website via HTTP communication. For asynchronous processing, standard Kotlin coroutines are used, performing communication processing on worker threads without blocking the UI thread. Information sent and received during communication with the Web server includes image URLs and image data, none of which are considered sensitive information. Therefore, the received image data may be attack data prepared by an attacker. For simplicity, the sample code does not implement judgment or countermeasures regarding whether received data is attack data. For similar reasons, exception handling for exceptions that may occur during HTTP connections and when displaying image data is kept to a minimum. Note that appropriate processing must be implemented according to the app's specifications.

In traditional Android, a method of setting the usesCleartextTraffic attribute in AndroidManifest.xml to true was used to uniformly permit HTTP communication for the entire app. However, in Android 17 and later, the usesCleartextTraffic attribute is deprecated and virtually invalidated. In Android 17 or higher environments, even if this attribute is set to true, all HTTP communication is blocked at the OS level unless there is explicit domain permission via Network Security Configuration. Therefore, in future HTTP communication implementations, rather than relying on batch communication permission for the app as a whole, strict access control that individually permits only the minimum necessary domains (e.g., www.jssec.org) using a Network Security Configuration file becomes mandatory.

Note that in Android 7.0 (API Level 24) and later, settings in Network Security Configuration take precedence over the android:usesCleartextTraffic attribute, so migrating to the approach described in this article is strongly recommended.

Points:

1. Do not include sensitive information in transmitted data
2. Assume cases where received data is transmitted from an attacker
3. Do not use batch app permission via the usesCleartextTraffic attribute; specify Network Security Configuration (NSC)

```
AndroidManifest.xml
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="org.jssec.android.https.imagesearch" >

    <uses-permission android:name="android.permission.INTERNET" />
```

(continues on next page)

(continued from previous page)

```
    <!-- Removed usesCleartextTraffic="true" due to deprecation, and specified␣
↪networkSecurityConfig -->
    <application
        android:icon="@mipmap/ic_launcher"
        android:allowBackup="false"
        android:label="@string/app_name"
        android:networkSecurityConfig="@xml/network_security_config" > <!-- Point␣
↪3 -->

        <activity
            android:name=".ImageSearchActivity"
            android:label="@string/app_name"
            android:theme="@android:style/Theme.Light"
            android:exported="true" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>
</manifest>
```

network_security_config.xml

```
<?xml version="1.0" encoding="utf-8"?>
<network-security-config>
    <!-- Default configuration: Forcibly block unencrypted (HTTP) communication␣
↪across the entire app -->
    <base-config cleartextTrafficPermitted="false" />

    <!-- 1. For HTTP image search (Allowing HTTP retrieval from www.jssec.org as␣
↪an exception) -->
    <domain-config cleartextTrafficPermitted="true">
        <domain includeSubdomains="true">www.jssec.org</domain>
    </domain-config>

    <!-- 2. For testing HTTPS CT verification bypass (Specifying a public test␣
↪domain) -->
    <domain-config>
        <domain includeSubdomains="true">no-sct.badssl.com</domain>
        <!-- Point 5 Disable CT verification to temporarily allow communication␣
↪with certificates that do not contain SCT information -->
        <certificateTransparency enabled="false" />
    </domain-config>
</network-security-config>
```

HttpImageSearch.kt

```
/*
 * Copyright (C) 2012-2026 Japan Smartphone Security Association
 *
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 *
 *      http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
```

(continues on next page)

(continued from previous page)

```
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */

package org.jssec.android.https.imagesearch

import kotlinx.coroutines.Dispatchers
import kotlinx.coroutines.withContext
import java.io.BufferedInputStream
import java.io.ByteArrayOutputStream
import java.io.IOException
import java.net.HttpURLConnection
import java.net.MalformedURLException
import java.net.ProtocolException
import java.net.URL

class HttpImageSearch {

    // Execute communication processing on a worker thread (Dispatchers.IO)
    suspend fun doInBackground(imageUrl: String): ByteArray? =␣
↪withContext(Dispatchers.IO) {
        var con: HttpURLConnection? = null
        var responseArray: ByteArray? = null
        try {
            // Point 1 Do not include sensitive information in transmitted data
            // Be careful not to include secret information when assembling image␣
↪URLs, etc.
            con = connectUrl(imageUrl) ?: return@withContext null
            checkResponse(con)

            // Point 2 Assume cases where received data is transmitted from an␣
↪attacker
            // Judgment and processing when search results are data from an␣
↪attacker (e.g., malicious image decoding vulnerability attack) are omitted for␣
↪sample simplicity
            responseArray = getByteArray(con)
        } catch (e: IOException) {
            // Exception handling omitted
        } finally {
            con?.disconnect()
        }
        return@withContext responseArray
    }

    private fun connectUrl(strUrl: String): HttpURLConnection? {
        var con: HttpURLConnection? = null
        try {
            val url = URL(strUrl)
            con = url.openConnection() as HttpURLConnection
            con.requestMethod = "GET"
            con.connect()
        } catch (e: ProtocolException) {
            // Exception handling omitted
        } catch (e: MalformedURLException) {
```

(continues on next page)

(continued from previous page)

```
            // Exception handling omitted
        } catch (e: IOException) {
            // Exception handling omitted
        }
        return con
    }

    private fun getByteArray(con: HttpURLConnection): ByteArray? {
        val buff = ByteArray(1024)
        var result: ByteArray? = null
        var inputStream: BufferedInputStream? = null
        var responseArray: ByteArrayOutputStream? = null
        try {
            inputStream = BufferedInputStream(con.inputStream)
            responseArray = ByteArrayOutputStream()
            var length: Int
            while (inputStream.read(buff).also { length = it } != -1) {
                if (length > 0) {
                    responseArray.write(buff, 0, length)
                }
            }
            result = responseArray.toByteArray()
        } catch (e: IOException) {
            e.printStackTrace()
        } finally {
            try {
                inputStream?.close()
            } catch (e: IOException) {
                // Exception handling omitted
            }
            try {
                responseArray?.close()
            } catch (e: IOException) {
                // Exception handling omitted
            }
        }
        return result
    }

    @Throws(IOException::class)
    private fun checkResponse(con: HttpURLConnection) {
        val statusCode = con.responseCode
        if (HttpURLConnection.HTTP_OK != statusCode) {
            throw IOException("HttpStatus: $statusCode")
        }
    }
}
```

ImageSearchActivity.kt

```
/*
 * Copyright (C) 2012-2026 Japan Smartphone Security Association
 *
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 *
```

(continues on next page)

(continued from previous page)

```
 *      http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */

package org.jssec.android.https.imagesearch

import android.graphics.Bitmap
import android.graphics.BitmapFactory
import android.os.Bundle
import android.view.View
import android.widget.EditText
import android.widget.ImageView
import android.widget.TextView
import androidx.activity.ComponentActivity
import androidx.lifecycle.lifecycleScope
import kotlinx.coroutines.Job
import kotlinx.coroutines.launch

class ImageSearchActivity : ComponentActivity() {

    private lateinit var mQueryBox: EditText
    private lateinit var mMsgBox: TextView
    private lateinit var mImgBox: ImageView
    private var mSearchJob: Job? = null

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)

        mQueryBox = findViewById(R.id.querybox)
        mMsgBox = findViewById(R.id.msgbox)
        mImgBox = findViewById(R.id.imageview)
    }

    override fun onPause() {
        // Cancel asynchronous processing when the Activity is destroyed or␣
↪transitions to the background
        mSearchJob?.cancel()
        super.onPause()
    }

    fun onHttpSearchClick(view: View) {
        val imageUrl = "http://www.jssec.org/common/images/main_visual_local.jpg"
        mMsgBox.text = imageUrl
        mImgBox.setImageBitmap(null)

        // Cancel in consideration of the possibility that the previous␣
↪asynchronous processing has not finished
        mSearchJob?.cancel()

        // Safely start asynchronous processing using lifecycleScope (without␣
```

(continues on next page)

(continued from previous page)

```
↪blocking the UI thread)
        mSearchJob = lifecycleScope.launch {
            val search = HttpImageSearch()
            val result = search.doInBackground(imageUrl)

            // Reflect results (automatically and safely executed on the UI thread)
            if (result == null) {
                mMsgBox.append("\nException occurred\n")
            } else {
                // Exception handling during image decoding and display is omitted␣
↪for sample simplicity
                val bmp: Bitmap? = BitmapFactory.decodeByteArray(result, 0, result.
↪size)
                mImgBox.setImageBitmap(bmp)
            }
        }
    }

    fun onHttpsSearchClick(view: View) {
        val query = mQueryBox.text.toString()
        mMsgBox.text = "HTTPS Search Started: $query"
        mImgBox.setImageBitmap(null)

        // Cancel if the previous search task is currently running
        mSearchJob?.cancel()

        // Safely start asynchronous HTTPS communication using lifecycleScope
        mSearchJob = lifecycleScope.launch {
            val search = HttpsImageSearch()
            val result = search.doInBackground(query)

            // Reflect results in the UI
            if (result == null) {
                mMsgBox.append("\nException occurred (communication failed)\n")
            } else {
                // Exception handling for image data decoding should be performed␣
↪appropriately according to app requirements
                val bmp: Bitmap? = BitmapFactory.decodeByteArray(result, 0, result.
↪size)
                mImgBox.setImageBitmap(bmp)
                mMsgBox.text = "HTTPS Search Completed: $query"
            }
        }
    }
}
```

5.4.1.2 Communicating via HTTPS

In HTTPS communication, not only is transmitted and received data encrypted, but verification is also performed to determine whether the connected server is authentic. For this purpose, during the handshake process at the start of HTTPS communication, verification is performed inside Android's HTTPS library on the server certificate sent from the server from the following perspectives:

- Must be a server certificate signed by a third-party Certificate Authority
- The server certificate's expiration date and other details must be valid

- The DNS name in the Subject's CN (Common Name) or SAN (Subject Alternative Names) of the server certificate must match the host name of the connected server

If these verifications fail, a server certificate verification error occurs. This error occurs when there is a deficiency in the server certificate or when an attacker is conducting a man-in-the-middle attack. Note that if an error occurs, appropriate processing must be executed according to the app's specifications.

For apps targeting Android 17 or higher, to further increase the security of HTTPS communication, the following robust security features are enabled by default at the OS level:

1. Default Enabling of ECH (Encrypted Client Hello)

In conventional TLS (HTTPS) communication, although the communication content itself was encrypted, the "SNI (Server Name Indication)" presenting the destination host name during handshake was transmitted in plaintext, creating a risk of third parties on the communication path snooping on the accessed domain name (= service being used). In Android 17 and later, "ECH," which encrypts this SNI portion as well, is enabled by default. This significantly lowers the risk of network monitors identifying the specific destination domain, greatly enhancing communication privacy. *For ECH to actually function, the app's communication library and the destination server/CDN must support the ECH protocol.

2. Default Verification of Certificate Transparency (CT)

To prevent man-in-the-middle attacks caused by misissuance or fraudulent issuance of TLS certificates, CT verification for publicly trusted TLS certificates is automatically applied by default in Android 17 and later. Through this, the destination certificate is recorded in public logs (CT logs), an SCT (Signed Certificate Timestamp) is attached as proof of log registration, and the client (such as Android) verifies this SCT. In short, it is a mechanism that checks whether the certificate is listed in public logs to detect unauthorized certificates. Attempting to connect to a server using a certificate that does not contain an SCT (not CT-compliant) results in an SSLHandshakeException during handshake and the communication is forcibly blocked. *Certificates acquired from public CAs in production environments (such as Let's Encrypt or DigiCert) pass CT without issue, but when using self-signed certificates in development or staging environments, a connection error occurs because they are not registered in CT logs. In this case, as described below, it is necessary to exceptionally disable CT verification (enabled="false") for the target domain only using Network Security Configuration.

Points:

1. Start URIs with https://
2. Sensitive information may be included in transmitted data
3. Verify the safety of received data even if it is data from a server connected via HTTPS
4. Perform appropriate exception handling for the app in response to SSLException
5. When using certificates without SCT or self-signed certificates in development and test environments, disable CT verification in Network Security Configuration (in the case of self-signed, additionally add trust settings for private CAs)

network_security_config.xml

```
<?xml version="1.0" encoding="utf-8"?>
<network-security-config>
    <!-- Default configuration: Forcibly block unencrypted (HTTP) communication␣
↪across the entire app -->
    <base-config cleartextTrafficPermitted="false" />

    <!-- 1. For HTTP image search (Allowing HTTP retrieval from www.jssec.org as␣
↪an exception) -->
    <domain-config cleartextTrafficPermitted="true">
        <domain includeSubdomains="true">www.jssec.org</domain>
    </domain-config>

    <!-- 2. For testing HTTPS CT verification bypass (Specifying a public test␣
↪domain) -->
```

(continues on next page)

(continued from previous page)

```
    <domain-config>
        <domain includeSubdomains="true">no-sct.badssl.com</domain>
        <!-- Point 5 Disable CT verification to temporarily allow communication␣
↪with certificates that do not contain SCT information -->
        <certificateTransparency enabled="false" />
    </domain-config>
</network-security-config>
```

HttpsImageSearch.kt

```
/*
 * Copyright (C) 2012-2026 Japan Smartphone Security Association
 *
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 *
 *      http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */


package org.jssec.android.https.imagesearch

import kotlinx.coroutines.Dispatchers
import kotlinx.coroutines.withContext
import java.io.BufferedInputStream
import java.io.ByteArrayOutputStream
import java.io.IOException
import java.net.MalformedURLException
import java.net.ProtocolException
import java.net.URL
import javax.net.ssl.HttpsURLConnection
import javax.net.ssl.SSLException

class HttpsImageSearch {

    // Safely execute HTTPS communication on a worker thread (Dispatchers.IO)
    suspend fun doInBackground(query: String): ByteArray? =␣
↪withContext(Dispatchers.IO) {
        var con: HttpsURLConnection? = null
        var responseArray: ByteArray? = null
        try {
            // 1st communication: Execute image search
            // Point 1 Start URIs with https://
            // Point 2 Because it is encrypted, sensitive information such as␣
↪passwords and tokens may be included in transmitted data
            val searchUrl = "https://www.jssec.org/"
            con = connectUrl(searchUrl) ?: return@withContext null
            checkResponse(con)

            // Process to retrieve image URL from search results (Mock URL used␣
```

(continues on next page)

(continued from previous page)

```
↪for sample simplicity)
            val imageUrl = "https://www.jssec.org/common/images/main_visual_local.
↪jpg"
            con.disconnect()

            // 2nd communication: Retrieve image data
            con = connectUrl(imageUrl) ?: return@withContext null
            checkResponse(con)

            // Point 3 Even when using HTTPS, confirm the safety of received data␣
↪(images) during decoding
            responseArray = getByteArray(con)

        } catch (e: SSLException) {
            // Point 4 Exception handling when a certificate verification error␣
↪(man-in-the-middle attack, expiration, etc.) occurs
            // In actual commercial apps, appropriately implement processing such␣
↪as prompting the user with a warning dialog
            e.printStackTrace()
        } catch (e: IOException) {
            // Other communication exception handling
            e.printStackTrace()
        } finally {
            con?.disconnect()
        }
        return@withContext responseArray
    }

    private fun connectUrl(strUrl: String): HttpsURLConnection? {
        var con: HttpsURLConnection? = null
        try {
            val url = URL(strUrl)
            con = url.openConnection() as HttpsURLConnection
            con.requestMethod = "GET"
            con.connect()
        } catch (e: ProtocolException) {
            // Exception handling omitted
        } catch (e: MalformedURLException) {
            // Exception handling omitted
        } catch (e: IOException) {
            // Exception handling omitted
        }
        return con
    }

    private fun getByteArray(con: HttpsURLConnection): ByteArray? {
        val buff = ByteArray(1024)
        var result: ByteArray? = null
        var inputStream: BufferedInputStream? = null
        var responseArray: ByteArrayOutputStream? = null
        try {
            inputStream = BufferedInputStream(con.inputStream)
            responseArray = ByteArrayOutputStream()
            var length: Int
            while (inputStream.read(buff).also { length = it } != -1) {
                if (length > 0) {
```

(continues on next page)

(continued from previous page)

```
                    responseArray.write(buff, 0, length)
                }
            }
            result = responseArray.toByteArray()
        } catch (e: IOException) {
            e.printStackTrace()
        } finally {
            try {
                inputStream?.close()
            } catch (e: IOException) {
                // Exception handling omitted
            }
            try {
                responseArray?.close()
            } catch (e: IOException) {
                // Exception handling omitted
            }
        }
        return result
    }

    @Throws(IOException::class)
    private fun checkResponse(con: HttpsURLConnection) {
        val statusCode = con.responseCode
        if (HttpsURLConnection.HTTP_OK != statusCode) {
            throw IOException("HttpStatus: $statusCode")
        }
    }
}
```

5.4.1.3 Communicating via HTTPS with private certificate

This section shows a sample code of HTTPS communication with a server certificate issued privately (private certificate), but not with that issued by a trusted third party authority. Please refer to "5.4.3.1. *How to Create Private Certificate and Configure Server Settings*" for creating a root certificate of a private certificate authority and private certificates and setting HTTPS settings in a Web server. The sample program has a cacert.crt file in assets. It is a root certificate file of private certificate authority.

The following sample code shows an application which gets an image on a Web server and shows it. HTTPS is used for the communication with the server. The worker thread for communication process using AsyncTask is created to avoid the communications performing on the UI thread. All contents (the URL of the image and the image data) sent/received in the communications with the server are considered as sensitive here. To show the sample code simply, no special handling for SSLException is performed. It is necessary to handle the exceptions properly depending on the application specifications.

Points:

1. Verify a server certificate with the root certificate of a private certificate authority.
2. URI starts with https://.
3. Sensitive information may be contained in send data.
4. Received data can be trusted as same as the server.
5. SSLException should be handled with an appropriate sequence in an application.

```
PrivateCertificateHttpsGet.java
/*
```

(continues on next page)

(continued from previous page)

```
 * Copyright (C) 2012-2026 Japan Smartphone Security Association
 *
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 *
 *      http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */

package org.jssec.android.https.privatecertificate;

import java.io.BufferedInputStream;
import java.io.ByteArrayOutputStream;
import java.io.IOException;
import java.net.HttpURLConnection;
import java.net.URL;
import java.security.KeyStore;
import java.security.SecureRandom;

import javax.net.ssl.HostnameVerifier;
import javax.net.ssl.HttpsURLConnection;
import javax.net.ssl.SSLContext;
import javax.net.ssl.SSLException;
import javax.net.ssl.SSLSession;
import javax.net.ssl.TrustManagerFactory;

import android.content.Context;
import android.os.AsyncTask;

public abstract class PrivateCertificateHttpsGet
        extends AsyncTask<String, Void, Object> {

    private Context mContext;

    public PrivateCertificateHttpsGet(Context context) {
        mContext = context;
    }

    @Override
    protected Object doInBackground(String... params) {
        TrustManagerFactory trustManager;
        BufferedInputStream inputStream = null;
        ByteArrayOutputStream responseArray = null;
        byte[] buff = new byte[1024];
        int length;

        try {
            URL url = new URL(params[0]);
            // *** POINT 1 *** Verify a server certificate with the root
            // certificate of a private certificate authority.
```

(continues on next page)

(continued from previous page)

```
            // Set keystore which includes only private certificate that is stored
            // in assets, to client.
            KeyStore ks = KeyStoreUtil.getEmptyKeyStore();
            KeyStoreUtil.loadX509Certificate(ks,
                    mContext.getResources().getAssets().open("cacert.crt"));

            // Verify host name
            HttpsURLConnection.setDefaultHostnameVerifier(new HostnameVerifier() {
                    @Override
                    public boolean verify(String hostname, SSLSession session) {
                        if (!hostname.equals(session.getPeerHost())) {
                            return false;
                        }
                        return true;
                    }
                });

            // *** POINT 2 *** URI starts with https://.
            // *** POINT 3 *** Sensitive information may be contained in send data.
            trustManager = TrustManagerFactory.getInstance(TrustManagerFactory.
↪getDefaultAlgorithm());
            trustManager.init(ks);
            SSLContext sslCon = SSLContext.getInstance("TLS");
            sslCon.init(null, trustManager.getTrustManagers(), new SecureRandom());

            HttpURLConnection con = (HttpURLConnection)url.openConnection();
            HttpsURLConnection response = (HttpsURLConnection)con;
            response.setDefaultSSLSocketFactory(sslCon.getSocketFactory());

            response.setSSLSocketFactory(sslCon.getSocketFactory());
            checkResponse(response);

            // *** POINT 4 *** Received data can be trusted as same as the server.
            inputStream = new BufferedInputStream(response.getInputStream());
            responseArray = new ByteArrayOutputStream();
            while ((length = inputStream.read(buff)) != -1) {
                if (length > 0) {
                    responseArray.write(buff, 0, length);
                }
            }
            return responseArray.toByteArray();
        } catch(SSLException e) {
            // *** POINT 5 *** SSLException should be handled with an appropriate
            // sequence in an application.
            // Exception process is omitted here since it's sample.
            return e;
        } catch(Exception e) {
            return e;
        } finally {
            if (inputStream != null) {
                try {
                    inputStream.close();
                } catch (Exception e) {
                    // This is sample, so omit the exception process
                }
            }
```

(continues on next page)

(continued from previous page)

```
            if (responseArray != null) {
                try {
                    responseArray.close();
                } catch (Exception e) {
                    // This is sample, so omit the exception process
                }
            }
        }
    }

    private void checkResponse(HttpURLConnection response) throws IOException {
        int statusCode = response.getResponseCode();
        if (HttpURLConnection.HTTP_OK != statusCode) {
            throw new IOException("HttpStatus: " + statusCode);
        }
    }
}
```

KeyStoreUtil.java

```
package org.jssec.android.https.privatecertificate;

import java.io.IOException;
import java.io.InputStream;
import java.security.KeyStore;
import java.security.KeyStoreException;
import java.security.NoSuchAlgorithmException;
import java.security.cert.Certificate;
import java.security.cert.CertificateException;
import java.security.cert.CertificateFactory;
import java.security.cert.X509Certificate;
import java.util.Enumeration;

public class KeyStoreUtil {
    public static KeyStore getEmptyKeyStore() throws KeyStoreException,
                                                     NoSuchAlgorithmException,
                                                     CertificateException,
                                                     IOException {
        KeyStore ks = KeyStore.getInstance("BKS");
        ks.load(null);
        return ks;
    }

    public static void loadAndroidCAStore(KeyStore ks)
        throws KeyStoreException, NoSuchAlgorithmException,
               CertificateException, IOException {
        KeyStore aks = KeyStore.getInstance("AndroidCAStore");
        aks.load(null);
        Enumeration<String> aliases = aks.aliases();
        while (aliases.hasMoreElements()) {
            String alias = aliases.nextElement();
            Certificate cert = aks.getCertificate(alias);
            ks.setCertificateEntry(alias, cert);
        }
    }

    public static void loadX509Certificate(KeyStore ks, InputStream is)
```

(continues on next page)

(continued from previous page)

```
        throws CertificateException, KeyStoreException {
        try {
            CertificateFactory factory = CertificateFactory.getInstance("X509");
            X509Certificate x509 =
                (X509Certificate)factory.generateCertificate(is);
            String alias = x509.getSubjectDN().getName();
            ks.setCertificateEntry(alias, x509);
        } finally {
            try {
                is.close();
            } catch (IOException e) {
                /* This is sample, so omit the exception process */
            }
        }
    }
}
```

PrivateCertificateHttpsActivity.java

```
/*
 * Copyright (C) 2012-2026 Japan Smartphone Security Association
 *
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 *
 *      http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */

package org.jssec.android.https.privatecertificate;

import android.app.Activity;
import android.graphics.Bitmap;
import android.graphics.BitmapFactory;
import android.os.AsyncTask;
import android.os.Bundle;
import android.view.View;
import android.widget.EditText;
import android.widget.ImageView;
import android.widget.TextView;

public class PrivateCertificateHttpsActivity extends Activity {

    private EditText mUrlBox;
    private TextView mMsgBox;
    private ImageView mImgBox;
    private AsyncTask<String, Void, Object> mAsyncTask ;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
```

(continues on next page)

(continued from previous page)

```
        setContentView(R.layout.activity_main);

        mUrlBox = (EditText)findViewById(R.id.urlbox);
        mMsgBox = (TextView)findViewById(R.id.msgbox);
        mImgBox = (ImageView)findViewById(R.id.imageview);
    }

    @Override
    protected void onPause() {
        // After this, Activity may be discarded, so cancel asynchronous process
        // in advance.
        if (mAsyncTask != null) mAsyncTask.cancel(true);
        super.onPause();
    }

    public void onClick(View view) {
        String url = mUrlBox.getText().toString();
        mMsgBox.setText(url);
        mImgBox.setImageBitmap(null);

        // Cancel, since the last asynchronous process might have not been
        // finished yet.
        if (mAsyncTask != null) mAsyncTask.cancel(true);

        // Since cannot communicate through UI thread, communicate by worker
        // thread by AsynchTask.
        mAsyncTask = new PrivateCertificateHttpsGet(this) {
                @Override
                protected void onPostExecute(Object result) {
                    // Process the communication result through UI thread.
                    if (result instanceof Exception) {
                        Exception e = (Exception)result;
                        mMsgBox.append("\nException occurs\n" + e.toString());
                    } else {
                        byte[] data = (byte[])result;
                        Bitmap bmp =
                            BitmapFactory.decodeByteArray(data, 0, data.length);
                        mImgBox.setImageBitmap(bmp);
                    }
                }
            }.execute(url);  // Pass URL and start asynchronization process
    }
}
```

5.4.2 Rule Book

When communicating via HTTP or HTTPS, adhere to the following rules.

1. *Send and Receive Sensitive Information via HTTPS Communication (Required)*
2. *Verify Safety of Received Data in HTTP Communication (Required)*
3. *Perform Appropriate Exception Handling Such As Notifying the User for SSLException (Required)*
4. *Do Not Create Custom TrustManagers (Required)*
5. *Do Not Create Custom HostnameVerifiers (Required)*

6. *When Communicating via HTTP, Do Not Use Batch Permission via the usesCleartextTraffic Attribute; Individually Permit Only Required Domains in Network Security Configuration (Required)*
7. *When Using Certificates Without SCT Information or Self-Signed Certificates in Development/Test Environments, Safely Opt Out of CT Verification in Network Security Configuration (Required)*

5.4.2.1 Send and Receive Sensitive Information via HTTPS Communication (Required)

In communication using HTTP, there is a risk of eavesdropping or tampering with sent and received information, or spoofing of the connected server. Send and receive sensitive information via HTTPS communication.

5.4.2.2 Verify Safety of Received Data in HTTP Communication (Required)

Because received data in HTTP communication can be controlled by an attacker, there is a possibility of receiving attack data targeting code vulnerabilities. Assuming that data of any value and format may be received, care must be taken in coding so that there are no vulnerabilities in the code processing received data. Also, regarding received data in HTTPS communication, received data must not be unconditionally considered safe. This is because the HTTPS connected server may be one prepared by an attacker, or the received data may be data generated in a location different from the connected server. See also "3.2. *Handling Input Data Carefully and Securely*".

5.4.2.3 Perform Appropriate Exception Handling Such As Notifying the User for SSLException (Required)

In HTTPS communication, an SSLException may occur during server certificate verification. An SSLException occurs due to a deficiency in the server certificate. Because a certificate deficiency may be caused by a man-in-the-middle attack by an attacker, implementing appropriate exception handling for SSLException is necessary. As examples of exception handling, notifying the user of communication failure due to SSLException or logging it can be considered. On the other hand, depending on the app, special notification to the user may not be required. In this way, because processing to be implemented differs depending on app specifications and characteristics, a decision must be made after thoroughly considering them.

In addition, because a man-in-the-middle attack may be taking place when an SSLException occurs, do not implement re-attempts to send or receive sensitive information via unencrypted communication such as HTTP.

5.4.2.4 Do Not Create Custom TrustManagers (Required)

To perform HTTPS communication with private certificates such as self-signed certificates, it is sufficient simply to change the KeyStore used for server certificate verification. However, as described in "5.4.3.3. *Risky Code that Disables Certificate Verification*", many sample codes available on the Internet introduce examples implementing dangerous TrustManagers. Apps implemented with reference to these samples may introduce vulnerabilities.

When wishing to perform HTTPS communication with private certificates, refer to the safe sample code in "5.4.1.3. *Communicating via HTTPS with private certificate*".

Normally, it is possible to implement custom TrustManagers safely; however, because there is a risk of introducing mistakes unless engineers have sufficient knowledge of cryptographic processing and cryptographic communication, this rule was deliberately made required.

5.4.2.5 Do Not Create Custom HostnameVerifiers (Required)

To perform HTTPS communication with private certificates such as self-signed certificates, it is sufficient simply to change the KeyStore used for server certificate verification. However, as described in "5.4.3.3. *Risky Code that Disables Certificate Verification*", many sample codes available on the Internet introduce examples using dangerous HostnameVerifiers. Apps implemented with reference to these samples may introduce vulnerabilities.

When wishing to perform HTTPS communication with private certificates, refer to the safe sample code in "5.4.1.3. *Communicating via HTTPS with private certificate*".

Normally, it is possible to implement custom HostnameVerifiers safely; however, because there is a risk of introducing mistakes unless engineers have sufficient knowledge of cryptographic processing and cryptographic communication, this rule was deliberately made required.

5.4.2.6 When Communicating via HTTP, Do Not Use Batch Permission via the usesCleartextTraffic Attribute; Individually Permit Only Required Domains in Network Security Configuration (Required)

In Android 17 and later, the usesCleartextTraffic attribute in AndroidManifest.xml is deprecated and virtually invalidated.

Even if this attribute is set to true, all HTTP communication is blocked at the OS level unless there is explicit domain permission via a Network Security Configuration file.

Therefore, when HTTP communication is required, rather than relying on batch permission for the app, you must individually permit only the minimum necessary domains that unavoidably perform cleartext communication (cleartextTrafficPermitted="true") using a Network Security Configuration file.

5.4.2.7 When Using Certificates Without SCT Information or Self-Signed Certificates in Development/Test Environments, Safely Opt Out of CT Verification in Network Security Configuration (Required)

In apps targeting Android 17 or higher, automatic verification of "Certificate Transparency (CT)" is enabled by default during HTTPS communication.

Certificates acquired from public CAs in production environments (such as Let's Encrypt) pass without issue, but attempting to connect to a server using a certificate that does not contain SCT information or a self-signed certificate, such as a staging server built for verification purposes, results in an SSLHandshakeException and the communication is forcibly blocked.

For this reason, limited to specific communication destinations for development and testing, it is necessary to specify the target domain in Network Security Configuration and exceptionally disable CT verification (<certificateTransparency enabled="false" />).

Note that when using a self-signed certificate, in addition to disabling CT verification, settings to trust the private CA certificate itself in the app (<trust-anchors>) must be written simultaneously.

5.4.3 Advanced Topics

5.4.3.1 How to Create Private Certificate and Configure Server Settings

In this section, how to create a private certificate and configure server settings in Linux such as Ubuntu and CentOS is described. Private certificate means a server certificate which is issued privately and is told from server certificates issued by trusted third party certificate authorities like Cybertrust and VeriSign.

Create private certificate authority

First of all, you need to create a private certificate authority to issue a private certificate. Private certificate authority means a certificate authority which is created privately as well as private certificate. You can issue plural private certificates by using the single private certificate authority. PC in which the private certificate authority is stored should be limited strictly to be accessed just by trusted persons.

To create a private certificate authority, you have to create two files such as the following shell script newca.sh and the setting file openssl.cnf and then execute them. In the shell script, CASTART and CAEND stand for the valid period of certificate authority and CASUBJ stands for the name of certificate authority. So these values need to be changed according to a certificate authority you create. While executing the shell script, the password for accessing the certificate authority is asked for 3 times in total, so you need to input it every time.

```
newca.sh -- Shell Script to create certificate authority
#!/bin/bash

umask 0077

CONFIG=openssl.cnf
CATOP=./CA
CAKEY=cakey.pem
CAREQ=careq.pem
CACERT=cacert.pem
CAX509=cacert.crt
CASTART=130101000000Z    # 2013/01/01 00:00:00 GMT
CAEND=230101000000Z      # 2023/01/01 00:00:00 GMT
CASUBJ="/CN=JSSEC Private CA/O=JSSEC/ST=Tokyo/C=JP"

mkdir -p ${CATOP}
mkdir -p ${CATOP}/certs
mkdir -p ${CATOP}/crl
mkdir -p ${CATOP}/newcerts
mkdir -p ${CATOP}/private
touch ${CATOP}/index.txt

openssl req -new -newkey rsa:2048 -sha256 -subj "${CASUBJ}" \
        -keyout ${CATOP}/private/${CAKEY} -out ${CATOP}/${CAREQ}
openssl ca -selfsign -md sha256 -create_serial -batch \
        -keyfile ${CATOP}/private/${CAKEY} \
        -startdate ${CASTART} -enddate ${CAEND} -extensions v3_ca \
        -in ${CATOP}/${CAREQ} -out ${CATOP}/${CACERT} \
        -config ${CONFIG}
openssl x509 -in ${CATOP}/${CACERT} -outform DER -out ${CATOP}/${CAX509}
```

```
openssl.cnf - Setting file of openssl command which 2 shell scripts refers in␣
↪common
[ ca ]
default_ca      = CA_default            # The default ca section

[ CA_default ]
dir             = ./CA                  # Where everything is kept
certs           = $dir/certs            # Where the issued certs are kept
crl_dir         = $dir/crl              # Where the issued crl are kept
database        = $dir/index.txt        # database index file.
#unique_subject = no                    # Set to 'no' to allow creation of␣
↪several ctificates with same subject.
new_certs_dir   = $dir/newcerts         # default place for new certs.
certificate     = $dir/cacert.pem       # The CA certificate
serial          = $dir/serial           # The current serial number
crlnumber       = $dir/crlnumber        # the current crl number must be␣
↪commented out to leave a V1 CRL
crl             = $dir/crl.pem          # The current CRL
private_key     = $dir/private/cakey.pem # The private key
RANDFILE        = $dir/private/.rand    # private random number file
x509_extensions = usr_cert              # The extentions to add to the cert
name_opt        = ca_default            # Subject Name options
cert_opt        = ca_default            # Certificate field options
policy          = policy_match
```

(continues on next page)

(continued from previous page)

```
[ policy_match ]
countryName             = match
stateOrProvinceName     = match
organizationName        = supplied
organizationalUnitName  = optional
commonName              = supplied
emailAddress            = optional

[ usr_cert ]
basicConstraints        = CA:FALSE
nsComment               = "OpenSSL Generated Certificate"
subjectKeyIdentifier    = hash
authorityKeyIdentifier  = keyid,issuer
subjectAltName          = @alt_names

[ v3_ca ]
subjectKeyIdentifier    = hash
authorityKeyIdentifier  = keyid:always,issuer
basicConstraints        = CA:true

[ alt_names ]
DNS.1                   = ${ENV::HOSTNAME}
DNS.2                   = *.${ENV::HOSTNAME}
```

After executing the above shall script, a directory named CA is created just under the work directory. This CA directory is just a private certificate authority. CA/cacert.crt file is the root certificate of the private certificate authority. And it's stored in assets directory of an application as described in "5.4.1.3. *Communicating via HTTPS with private certificate*", or it's installed in Android device as described in "5.4.3.2. *Install Root Certificate of Private Certificate Authority to Android OS's Certification Store*".

Create private certificate

To create a private certificate, you have to create a shell script like the following newca.sh and execute it. In the shell script, SVSTART and SVEND stand for the valid period of private certificate, and SVSUBJ stands for the name of Web server, so these values need to be changed according to the target Web server. Especially, you need to make sure not to set a wrong host name to /CN of SVSUBJ with which the host name of Web server is to be specified. While executing the shell script, the password for accessing the certificate authority is asked, so you need to input the password which you have set when creating the private certificate authority. After that, y/n is asked 2 times in total and you need to input y every time.

```
newsv.sh - Shell script which issues private certificate
#!/bin/bash

umask 0077

CONFIG=openssl.cnf
CATOP=./CA
CAKEY=cakey.pem
CACERT=cacert.pem
SVKEY=svkey.pem
SVREQ=svreq.pem
SVCERT=svcert.pem
SVX509=svcert.crt
SVSTART=130101000000Z    # 2013/01/01 00:00:00 GMT
SVEND=230101000000Z      # 2023/01/01 00:00:00 GMT
HOSTNAME=selfsigned.jssec.org
SVSUBJ="/CN="${HOSTNAME}"/O=JSSEC Secure Coding Group/ST=Tokyo/C=JP"
```

(continues on next page)

(continued from previous page)

```
openssl genrsa -out ${SVKEY} 2048
openssl req -new -key ${SVKEY} -subj "${SVSUBJ}" -out ${SVREQ}
openssl ca -md sha256 \
        -keyfile ${CATOP}/private/${CAKEY} -cert ${CATOP}/${CACERT} \
        -startdate ${SVSTART} -enddate ${SVEND} \
        -in ${SVREQ} -out ${SVCERT} -config ${CONFIG}
openssl x509 -in ${SVCERT} -outform DER -out ${SVX509}
```

After executing the above shall script, a private key file for Web server “svkey.pem” and private certificate file “svcert.pem” are created just under the work directory.

If the Web server is Apache, you will specify prikey.pem and cert.pem in the configuration file as follows

```
SSLCertificateFile "/path/to/svcert.pem"
SSLCertificateKeyFile "/path/to/svkey.pem"
```

5.4.3.2 Install Root Certificate of Private Certificate Authority to Android OS's Certification Store

In the sample code of "5.4.1.3. *Communicating via HTTPS with private certificate*", the method to establish HTTPS sessions to a Web server from one application using a private certificate by installing the root certificate into the application is introduced. In this section, the method to establish HTTPS sessions to Web servers from all applications using private certificates by installing the root certificate into Android OS is to be introduced. Note that all you install should be certificates issued by trusted certificate authorities including your own certificate authorities.

However, the method described here can be used in versions prior to Android 6.0 (API level 23) only. Starting from Android 7.0 (API level 24), even if the root certificate of the private certificate authority is installed, the system ignores it. Starting from API level 24, to use a private certificate, refer to the section “Communicating via HTTPS with private certificates” in "5.4.3.7. *Network Security Configuration*".

First of all, you need to copy the root certificate file "cacert.crt" to the internal storage of an Android device. You can also get the root certificate file used in the sample code from [https://www.jssec.org/dl/android_securecoding_sample_cacert.crt{]}(https://www.jssec.org/dl/android_securecoding_sample_cacert.crt).

And then, you will open Security page from Android Settings and you can install the root certificate in an Android device by doing as follows.

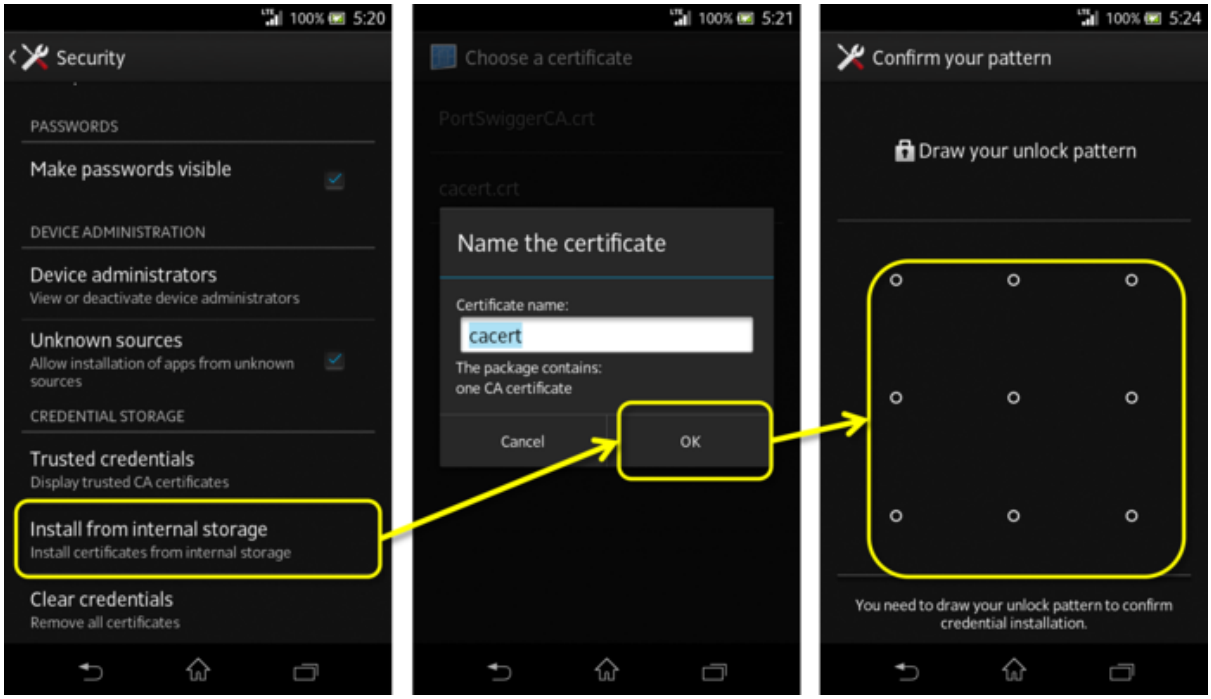


Fig. 5.4.2: Steps to install root certificate of private certificate authority

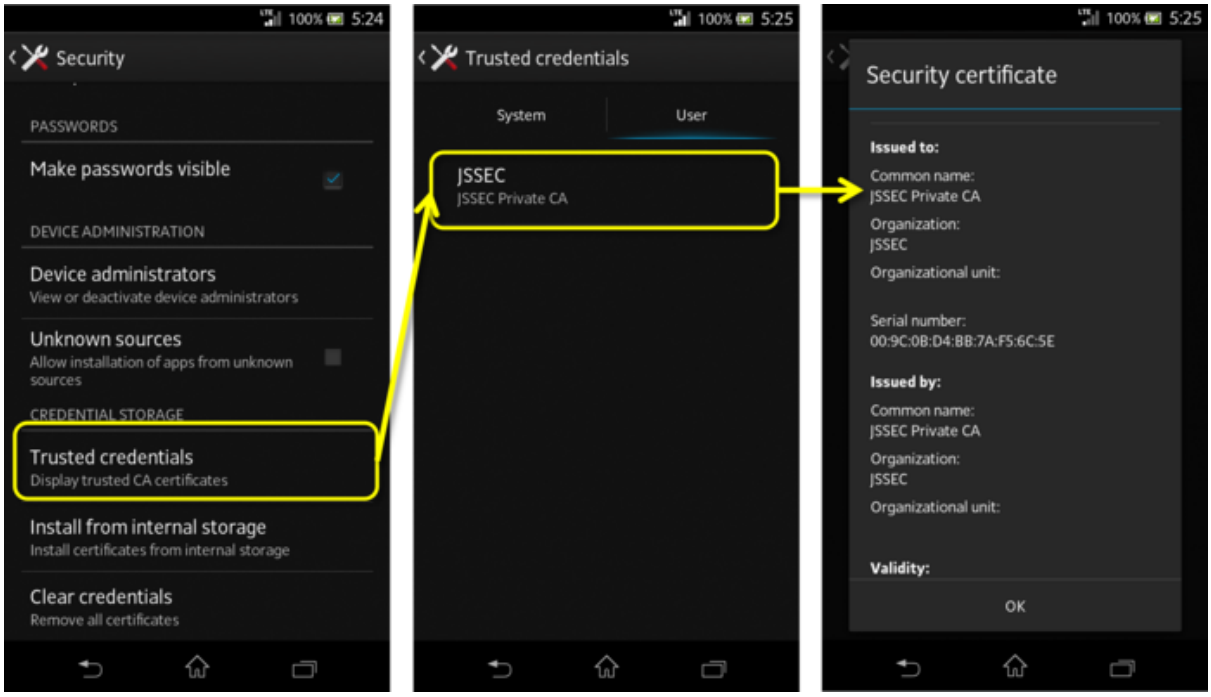


Fig. 5.4.3: Checking if root certificate is installed or not

Android Once the root certificate is installed in Android OS, all applications can correctly verify every private certificate issued by the certificate authority. The following figure shows an example when displaying https://selfsigned.jssec.org/droid_knight.png in Chrome browser.

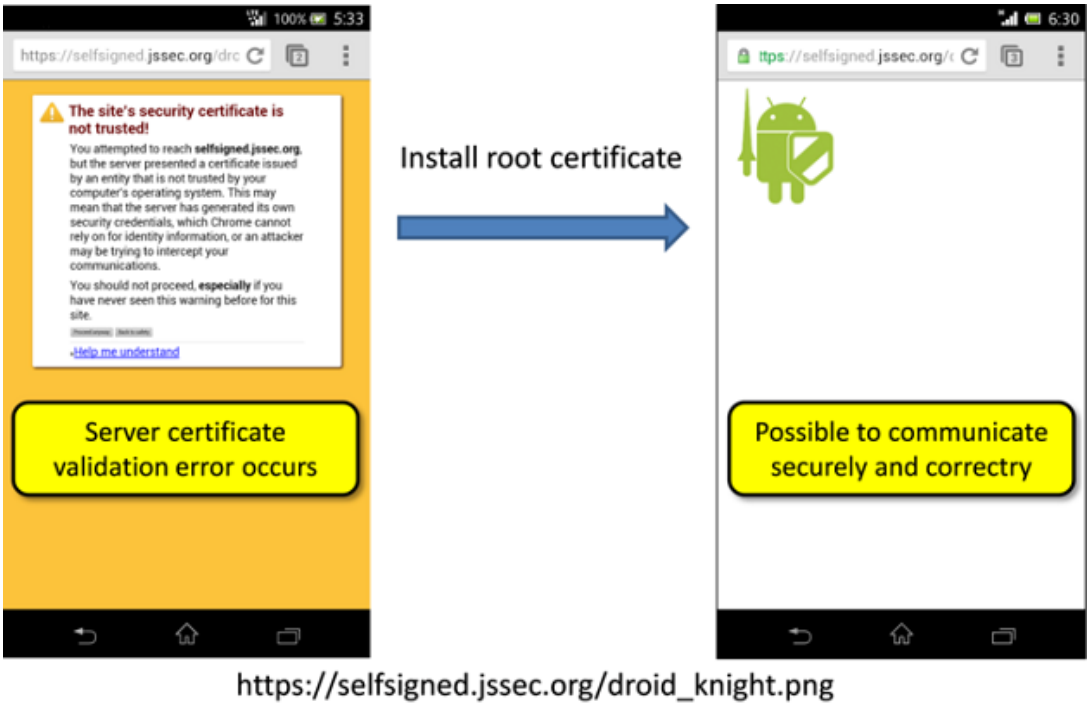


Fig. 5.4.4: Once root certificate installed, private certificates can be verified correctly

By installing the root certificate this way, even applications using the sample code "5.4.1.2. *Communicating via HTTPS*" can correctly connect via HTTPS to a Web server which is operated with a private certificate.

5.4.3.3 Risky Code that Disables Certificate Verification

A lot of incorrect samples (code snippets), which allow applications to continue to communicate via HTTPS with Web servers even after certificate verification errors occur, are found on the Internet. Since they are introduced as the way to communicate via HTTPS with a Web server using a private certificate, there have been so many applications created by developers who have used those sample codes by copy and paste. Unfortunately, most of them are vulnerable to man-in-the-middle attack. As mentioned in the top of this article, "In 2012, many defects in implementation of HTTPS communication were pointed out in Android applications", many Android applications which would have implemented such vulnerable codes have been reported.

Several code snippets to cause vulnerable HTTPS communication are shown below. When you find this type of code snippets, it's highly recommended to replace the sample code of "5.4.1.3. *Communicating via HTTPS with private certificate*".

Risk:Case which creates empty TrustManager

```
TrustManager tm = new X509TrustManager() {

    @Override
    public void checkClientTrusted(X509Certificate[] chain,
            String authType) throws CertificateException {
        // Do nothing -> accept any certificates
    }

    @Override
    public void checkServerTrusted(X509Certificate[] chain,
            String authType) throws CertificateException {
        // Do nothing -> accept any certificates
    }
```

(continues on next page)

(continued from previous page)

```
    @Override
    public X509Certificate[] getAcceptedIssuers() {
        return null;
    }
};
```

Risk:Case which creates empty HostnameVerifier

```
HostnameVerifier hv = new HostnameVerifier() {
    @Override
    public boolean verify(String hostname, SSLSession session) {
        // Always return true -> Accespt any host names
        return true;
    }
};
```

Risk:Case that ALLOW_ALL_HOSTNAME_VERIFIER is used.

```
SSLSocketFactory sf;
[...]
sf.setHostnameVerifier(SSLSocketFactory.ALLOW_ALL_HOSTNAME_VERIFIER);
```

5.4.3.4 A note regarding the configuration of HTTP request headers

If you wish to specify your own individual HTTP request header for HTTP or HTTPS communication, use the setRequestProperty() or addRequestProperty() methods in the URLConnection class. If you will be using input data received from external sources as parameters for these methods, you must implement HTTP header-injection protections. The first step in attacks based on HTTP header injection is to include carriage-return codes—which are used as separators in HTTP headers—in input data. For this reason, all carriage-return codes must be eliminated from input data.

Configure HTTP request header

```
public byte[] openConnection(String strUrl, String strLanguage, String strCookie) {
    // HttpURLConnection is a class derived from URLConnection
    HttpURLConnection connection;

    try {
        URL url = new URL(strUrl);
        connection = (HttpURLConnection) url.openConnection();
        connection.setRequestMethod("GET");

        // *** POINT *** When using input values in HTTP request
        // headers, check the input data in accordance with the
        // application's requirements(*)
        if (strLanguage.matches("^[a-zA-Z ,-]+$")) {
            connection.addRequestProperty("Accept-Language", strLanguage);
        } else {
            throw new IllegalArgumentException("Invalid Language : " +
                                               strLanguage);
        }
        // *** POINT *** Or URL-encode the input data
        // (as appropriate for the purposes of the app in queestion)
        connection.setRequestProperty("Cookie",
                                      URLEncoder.encode(strCookie, "UTF-8"));
```

(continues on next page)

(continued from previous page)

```
        connection.connect();

        [...]
```

* See "3.2. *Handling Input Data Carefully and Securely*".

5.4.3.5 Notes and sample implementations for pinning

When an app uses HTTPS communication, one step in the handshake procedure carried out at the start of the communication is to check whether or not the certificate sent from the remote server is signed by a third-party certificate authority. However, attackers may acquire improper certificates from third-party authentication agents, or may acquire signed keys from a certificate authority to construct improper certificates. In such cases, apps will be unable to detect the attack during the handshake process—even in the event of a lure to an improper server established by the attacker, or of an man-in-the-middle attack —and, as a result, there is a possibility that damage may be done.

"The technique of pinning" is an effective strategy for preventing man-in-the-middle attacks using these types of certificates from improper third-party certificate authorities. In this method, certificates and public keys for remote servers are stored in advance within an app, and this information is used for handshake processing and re-testing after handshake processing has completed.

Pinning may be used to restore the security of communications in cases where the credibility of a third-party certificate authority—the foundation of public-key infrastructure—has been tarnished. App developers should assess the asset level handled by their own apps and decide whether or not to implement these tests.

Use certificates and public keys stored within an app during the handshake procedure

To use information contained in remote-server certificates or public keys stored within an app during the handshake procedure, an app must create its own KeyStore containing this information and use it when communicating. This will allow the app to detect improprieties during the handshake procedure even in the event of a man-in-the-middle attack using a certificate from an improper third-party certificate authority, as described above. Consult the sample code presented in the section titled "5.4.1.3. *Communicating via HTTPS with private certificate*" for detailed methods of establishing your app's own KeyStore to conduct HTTPS communication.

Use certificates and public-key information stored within an app for re-testing after the handshake procedure is complete

To re-test the remote server after the handshake procedure has completed, an app first obtains the certificate chain that was tested and trusted by the system during the handshake, then compares this certificate chain against the information stored in advance within the app. If the result of this comparison indicates agreement with the information stored within the app, the communication may be permitted to proceed; otherwise, the communication procedure should be aborted.

However, if an app uses the methods listed below in an attempt to obtain the certificate chain that the system trusted during the handshake, the app may not obtain the expected certificate chain, posing a risk that the pinning may not function properly[12].

- javax.net.ssl.SSLSession.getPeerCertificates()
- javax.net.ssl.SSLSession.getPeerCertificateChain()

What these methods return is not the certificate chain that was trusted by the system during the handshake, but rather the very certificate chain that the app received from the communication partner itself. For this reason, even if an man-in-the-middle attack has resulted in a certificate from an improper certificate authority being appended to the certificate chain, the above methods will not return the certificate that was trusted by the system during the handshake; instead, the certificate of the server to which the app was originally attempting to connect will also be returned at the same time. This certificate—"the certificate of the server to which the app was originally attempting to connect"—will, because of pinning, be equivalent to the certificate pre-stored within the app; thus re-testing it will

[12] The following article explains this risk in detail: https://www.synopsys.com/blogs/software-security/ineffective-certificate-pinning-implementations/

not detect any improprieties. For this and other similar reasons, it is best to avoid using the above methods when implementing re-testing after the handshake.

On Android versions 4.2 (API Level 17) and later, using the checkServerTrusted() method within net.http.X509TrustManagerExtensions will allow the app to obtain only the certificate chain that was trusted by the system during the handshake.

An example illustrating pinning using X509TrustManagerExtensions

```
// Store the SHA-256 hash value of the public key included in the correct
// certificate for the remote server (pinning)
private static final Set<String> PINS = new HashSet<>(Arrays.asList(
    new String[] {
            "d9b1a68fceaa460ac492fb8452ce13bd8c78c6013f989b76f186b1cbba1315c1",
            "cd13bb83c426551c67fabcff38d4496e094d50a20c7c15e886c151deb8531cdc"
    }
));

// Communicate using AsyncTask work threads
protected Object doInBackground(String... strings) {

    [...]

    // Obtain the certificate chain that was trusted by the system by
    // testing during the handshake
    X509Certificate[] chain =
        (X509Certificate[]) connection.getServerCertificates();
    X509TrustManagerExtensions trustManagerExt =
        new X509TrustManagerExtensions(
            (X509TrustManager) (trustManagerFactory.getTrustManagers()[0]));
    List<X509Certificate> trustedChain =
        trustManagerExt.checkServerTrusted(chain, "RSA", url.getHost());

    // Use public-key pinning to test
    boolean isValidChain = false;
    for (X509Certificate cert : trustedChain) {
        PublicKey key = cert.getPublicKey();
        MessageDigest md = MessageDigest.getInstance("SHA-256");
        String keyHash = bytesToHex(md.digest(key.getEncoded()));

        // Compare to the hash value stored by pinning
        if(PINS.contains(keyHash)) isValidChain = true;
    }
    if (isValidChain) {
        // Proceed with operation
    } else {
        // Do not proceed with operation
    }

    [...]
}

private String bytesToHex(byte[] bytes) {
    StringBuilder sb = new StringBuilder();
    for (byte b : bytes) {
        String s = String.format("%02x", b);
        sb.append(s);
    }
    return sb.toString();
```

(continues on next page)

(continued from previous page)

```
}
```

5.4.3.6 Strategies for addressing OpenSSL vulnerabilities using Google Play Services

Google Play Services (version 5.0 and later) provides a framework known as Provider Installer. This may be used to address vulnerabilities in Security Provider, an implementation of OpenSSL and other encryption-related technologies. For details, see Section "5.6.3.4. *Addressing Vulnerabilities with Security Provider from Google Play Services*".

5.4.3.7 Network Security Configuration

Android 7.0 (API Level 24) introduced a framework known as Network Security Configuration that allows individual apps to configure their own security settings for network communication. Using this framework makes it easy for apps to incorporate a variety of techniques for improving app security, including not only HTTPS communication with private certificates and public key pinning but also prevention of unencrypted (HTTP) communication and the use of private certificates enabled only during debugging[13].

The various types of functionality offered by Network Security Configuration may be accessed simply by configuring settings in xml files, which may be applied to the entirety of an app's HTTP and HTTPS communications. This eliminates the need for modifying an app's code or carrying out any additional processing, simplifying implementation and providing an effective protection against Incorporating bugs or vulnerabilities.

Communicating via HTTPS with private certificates

Section "5.4.1.3. *Communicating via HTTPS with private certificate*" presents sample code that performs HTTPS communication with private certificates (e.g. self-signed certificates or intra-company certificates). However, by using Network Security Configuration, developers may use private certificates without implementation presented in the sample code of Section "5.4.1.2. *Communicating via HTTPS*".

Use private certificates to communicate with specific domains

```
<?xml version="1.0" encoding="utf-8"?>
<network-security-config>
    <domain-config>
        <domain includeSubdomains="true">jssec.org</domain>
        <trust-anchors>
            <certificates src="@raw/private_ca" />
        </trust-anchors>
    </domain-config>
</network-security-config>
```

In the example above, the private certificates (private_ca) used for communication may be stored as resources within the app, with the conditions for their use and their range of applicability described in .xml files. By using the <domain-config> tag, it is possible to apply private certificates to specific domains only. To use private certificates for all HTTPS communications performed by the app, use the <base-config> tag, as shown below.

Use private certificates for all HTTPS communications performed by the app

```
<?xml version="1.0" encoding="utf-8"?>
<network-security-config>
    <base-config>
        <trust-anchors>
            <certificates src="@raw/private_ca" />
        </trust-anchors>
    </base-config>
</network-security-config>
```

[13] For more information on Network Security Configuration, see https://developer.android.com/training/articles/security-config.html

Pinning

We mentioned public key pinning in Section "5.4.3.5. *Notes and sample implementations for pinning*" By using Network Security Configuration to configure settings as in the example below, you eliminate the need to implement the authentication process in your code; instead, the specifications in the xml file suffice to ensure proper authentication.

Use public key pinning for HTTPS communication

```
<?xml version="1.0" encoding="utf-8"?>
<network-security-config>
    <domain-config>
        <domain includeSubdomains="true">jssec.org</domain>
        <pin-set expiration="2018-12-31">
            <pin digest="SHA-256">e30Lky+iWK21yHSls5DJoRzNikOdvQUOGXvurPidc2E=</
↪pin>
            <!-- for backup -->
            <pin digest="SHA-256">fwza0LRMXouZHRC8Ei+4PyuldPDcf3UKgO/04cDM1oE=</
↪pin>
        </pin-set>
    </domain-config>
</network-security-config>
```

The quantity described by the <pin> tag above is the base64-encoded hash value of the public key used for pinning. The only supported hash function is SHA-256.

Prevent unencrypted (HTTP) communication

Using Network Security Configuration allows you to prevent HTTP communication (unencrypted communication) from apps.

The methods of restricting unencrypted communications are as follows.

1. Basically, the <base-config> tag is used to restrict unencrypted communications (HTTP communication) in communication with all domains[14]
2. Only for domains that require unencrypted communications for unavoidable reasons, the <domain-config> tag can be used to individually set exceptions that allow unencrypted communications. For details on determining whether unencrypted communications should be permitted, refer to "5.4.1.1. *Communicating via HTTP*".

Unencrypted communications are restricted by setting the cleartextTrafficPermitted attribute to false. An example of this is shown below.

```
<?xml version="1.0" encoding="utf-8"?>
<network-security-config>
    <!-- Disallow unencrypted communication by default -->
    <base-config cleartextTrafficPermitted="false">
    </base-config>
    <!-- Only for domains that require unencrypted communications for unavoidable␣
↪reason,
         use <domain-config> tag to individualy set to "true" -->
    <domain-config cleartextTrafficPermitted="true">
       <domain includeSubdomains="true">www.jssec.org</domain>
    </domain-config>
</network-security-config>
```

This setting is also applied in the WebView from Android 8.0 (API level 26), but be aware that it is not applied to WebView for Android 7.1 (API level 25) and earlier.

Prior to Android 9.0 (API level 28), the default value of the attribute cleartextTrafficPermitted was true, but from Android 9.0, it was changed to false. For this reason, if targeting API level 28 and higher, declaration using <base-

[14] See the following API reference about how the Network Security Configuration works for non-HTTP connections. https://developer.android.com/reference/android/security/NetworkSecurityPolicy.html#isCleartextTrafficPermitted

config> in the above example is not needed. However, to clearly define the intention and to avoid the effect of different behavior depending on the target API level, explicitly including as shown in the example above is recommended.

Private certificates exclusively for debugging purposes

For purposes of debugging during app development, developers may wish to use private certificates to communicate with certain HTTPS servers that exist for app-development purposes. In this case, developers must be careful to ensure that no dangerous implementations—including code that disables certificate authentication—are incorporated into the app; this is discussed in Section "5.4.3.3. *Risky Code that Disables Certificate Verification*". In Network Security Configuration, settings may be configured as in the example below to specify a set of certificates to be used only when debugging (only if android:debuggable is set to "true" in the file AndroidManifest.xml). This eliminates the risk that dangerous code may inadvertently be retained in the release version of an app, thus constituting a useful means of preventing vulnerabilities.

Use private certificates only when debugging

```
<?xml version="1.0" encoding="utf-8"?>
<network-security-config>
    <debug-overrides>
        <trust-anchors>
            <certificates src="@raw/private_cas" />
        </trust-anchors>
    </debug-overrides>
</network-security-config>
```

5.4.3.8 (Column): Transitioning to TLS1.2/TLS1.3 for secure connections

The 1994 release of SSL 2, which was its first public release, had a major vulnerability in security protocol, and so SSL 3.0 (RFC 6101) was completely redesigned from the ground up and was released in the second half of 1995. However, due to a vulnerability known as POODLE[15] announced by the Google Security Team in 2014, it was found that the padding for SSL 3.0 was not safe. In TLS 1.0 (RFC 2246), which was released in 1999, a defect in the padding design was corrected, but an attack method known as BEAST that extracts encrypted data was announced in 2011 by Duong and Rizzo[16]. TLS 1.1 (RFC 4346) was released in 2006 with security fixes (enhanced safety from TLS 1.0), and TLS 1.2 (RFC 5246), which was released in 2008, enables the use of even stronger encryption algorithms, including the use of SHA-2 hash functions (SHA-256 and SHA-384) and supports cipher suites where authenticated encryption with associated data (AEAD) usage modes (GCM, CCM) can be used.

With this as a background, in its guidelines[17] on TLS issued on October 15, 2018, the (U.S.) National Institute of Standards and Technology (NIST) either deprecated or prohibited the use of TLS 1.1 and lower, and it requires not only government agencies, but also the servers that support non-government apps to migrate to TLS 1.2. In line with this move, given the rash of security incidents in recent years and the availability of new TLS versions, an increasing number of sites and services are discontinuing support for “old versions of SSL or TLS”, and the transition to TLS 1.2 is well underway[18].

For example, one manifestation of this transition is a new security standard known as "the Payment Card Industry Data Security Standard (PCI DSS)", established by the Payment Card Industry Security Standards Council (PCI SSC). The latest version is v3.2.1 released in May 2018[19]. Smartphones and tablets are also widely used for E-commerce today, with credit cards typically used for payment. Indeed, we expect that many users of this document (Android Application Secure Design / Secure Coding Guide) will offer services that send credit-card information and other data to the server side; when using credit cards in networked environments, it is essential to ensure the security of the data pathway, and PCI DSS is a standard that governs the handling of member data in services of this type, designed with the objective of preventing improper card use, information leaks, and other harmful consequences. Among

[15] “This POODLE bites: exploiting the SSL 3.0 fallback”(Google Security Team, October 14, 2014) (https://googleonlinesecurity.blogspot.co.uk/2014/10/this-poodle-bites-exploiting-ssl-30.html)

[16] “Here come the ⊕ Ninjas”(Thai Duong, Juliano Rizzo, May 13, 2011) (http://www.hit.bme.hu/%7Ebuttyan/courses/EIT-SEC/abib/04-TLS/BEAST.pdf)

[17] “Guidelines for the Selection, Configuration, and Use of Transport Layer Security (TLS) Implementations” (Rvision 2, October 2018) (https://csrc.nist.gov/CSRC/media/Publications/sp/800-52/rev-2/draft/documents/sp800-52r2-draft2.pdf)

[18] Encryption Design Guidelines, IPA (https://www.ipa.go.jp/security/vuln/ssl_crypt_config.html)

[19] “Requirements and Security Assessment Procedures” (Version 3.2.1, May 2018) (https://ja.pcisecuritystandards.org/document_library)

these security standards, although the exact version numbers are not specified, support for all SSL versions and early TLS versions susceptible to known exploits (attack programs) was discontinued on June 30, 2018, and websites were required to upgrade to a safer version (TLS 1.2 or higher).

In communication between smartphones and servers, the need to ensure the security of data pathways is not restricted to handling of credit-card information, but is also an extremely important aspect of operations involving the handling of private data or other sensitive information. Thus, the need to transition to secure connections using TLS 1.2 on the service-provision (server) side may now be said to be an urgent requirement.

On the other hand, in Android—which runs on the client side—WebView functionality supporting TLS 1.1 and later versions has been available since Android 4.4 (Kitkat), and for direct HTTP communication since Android 4.1 (early Jelly Bean), although some additional implementation is needed in this case.

Among service developers, the adoption of TLS 1.2 means cutting off access to users of Android 4.3 and earlier versions, so it might seem that such a step would have significant repercussions. However, as shown in the figure below, the most recent data[20] (current as of May 2019) show that Android versions 4.4 and later account for the overwhelming majority—96.2%—of all Android systems currently in use. In view of this fact, and considering the importance of guaranteeing the security of assets handled by apps, we recommend that serious consideration be paid to transitioning to TLS 1.2.

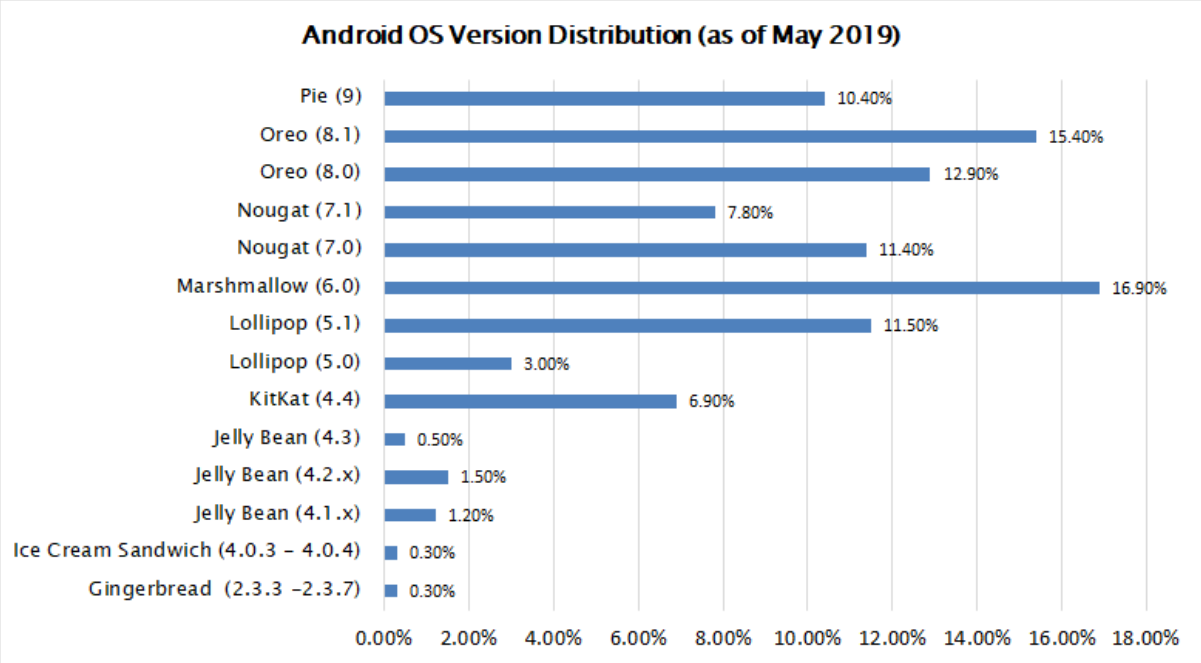


Fig. 5.4.5: Distribution of OS versions among Android systems in current use(Source: Android Developers site)

TLS 1.3 (RFC 8446), which was released in August 2018, was a complete redesign of protocols and encryption algorithms for the purpose of providing fixes for new vulnerabilities and exploits discovered since the issuing of TLS 1.2 and for providing performance enhancements[21]. Starting from Android 10, platform TLS implementation supports TLS 1.3, and TLS 1.3 is enabled for all TLS connections by default[22]. Also, the following cipher suites with low safety were removed starting from Android 10 (mode: CBC, MAC: SHA2)[23].

- TLS_RSA_WITH_AES_128_CBC_SHA256
- TLS_RSA_WITH_AES_256_CBC_SHA256
- TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256
- TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA384

[20] Distribution dashboard - Platform versions (https://developer.android.com/about/dashboards/index.html)
[21] The Transport Layer Security (TLS) Protocol Version 1.3 (https://datatracker.ietf.org/doc/rfc8446/)
[22] Android Q features and APIs - TLS 1.3 support (https://developer.android.com/preview/features#tls-1.3)
[23] SHA-2 CBC cipher suites removed (https://developer.android.com/preview/behavior-changes-all#sha2-cbc-cipher-suites)

- TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256
- TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384

5.4.3.9 Enabling ECH (Encrypted Client Hello)

For applications targeting Android 17 or higher, ECH (Encrypted Client Hello), a TLS extension that encrypts the Server Name Indication (SNI) during the TLS handshake, is enabled by default. Even in traditional HTTPS communications, the content of the communication was encrypted, but the SNI representing which site is being accessed (domain name) was transmitted in plaintext, which could allow the name of the service being used to be inferred from the domain name. With ECH in Android 17, this domain name part is also encrypted, preventing network monitors from easily identifying the specific domain the application is connecting to, thus further enhancing user privacy.

ECH is automatically utilized when establishing TLS connections within the application, but for ECH to actually function, the following conditions must be met:

- Network libraries supporting ECH: Supported by HttpEngine, WebView, OkHttp, etc.
- Servers supporting ECH: ECH configuration is present in HTTPS DNS (HTTPS RR), and the CDN (such as Cloudflare) or remote server being used supports the ECH protocol.
- Successful TLS negotiation: ECH is automatically used if the conditions are met. If not, it falls back to traditional TLS (if negotiation is not possible, a mechanism called ECH GREASE operates, where the client sends an ECH extension containing randomized content).

Table 5.4.3: Communication Differences with ECH Enabled

Item	Traditional TLS	Android 17 (ECH)
Communication Content	Encrypted	Encrypted
Domain (SNI)	Plaintext visible	Encrypted
Monitoring Resistance	△	◎

Impact on Debugging and Network Monitoring

If URL filtering is performed by security products or if proxies are used, the encryption of SNI may have an impact. Also, from the perspective of the development environment, since identifying the domain from communication logs (such as pcap) becomes impossible, there may be cases where debugging using packet capture is affected.

Customization using Network Security Configuration

ECH is enabled automatically, and basically no configuration is required by application developers. However, if you want to customize this behavior for debugging purposes or other reasons, use the Network Security Configuration file. In Android 17, a new <domainEncryption> element has been added, allowing you to select the ECH mode ("enabled", "disabled", etc.) globally or per domain within the <base-config> or <domain-config> tags.

```
<?xml version="1.0" encoding="utf-8"?>
<network-security-config>
    <domain-config>
        <domain includeSubdomains="true">example.com</domain>
        <domainEncryption mode="enabled" />
    </domain-config>
</network-security-config>
```

Since the key to making ECH work reliably depends on support not only on the application side but also on the server and DNS sides, it is desirable to verify the configuration of the entire network.

5.4.3.10 Certificate Transparency (CT) Compliance

For applications targeting Android 17 or higher, Certificate Transparency (CT) verification for publicly trusted TLS certificates is enabled by default. TLS certificates (certificates for HTTPS) are issued by CAs (Certificate Authorities), but misissuance and unauthorized issuance have occurred in the past. As a result, the risk of exposure to MITM attacks and other security threats has been considered a problem, and CT was introduced as a countermeasure. CT records issued certificates in public logs (CT logs) and attaches an SCT (Signed Certificate Timestamp) as proof of log registration, which clients (such as Android) then verify. In other words, it is a mechanism to detect unauthorized certificates by checking whether the certificate is listed in the public logs.

In Android 16 or earlier, CT was an opt-in feature that would not be used unless manually enabled, but in Android 17, it is enabled by default and applied implicitly, so verification runs without the developer having to be conscious of it. If a certificate is not CT-compliant, there is a possibility of TLS connection errors, API communication failures, or application crashes.

OS of Behavior Changes Verification Example

Whether CT verification is actually functioning can be confirmed by attempting to connect to a test server that does not include SCT (such as https://no-sct.badssl.com). In an environment targeting Android 17 (a pure network environment not passing through proxies, etc.), it can be confirmed that the communication is blocked and an SSLHandshakeException occurs, failing the SCT verification.

```
// Code to verify CT behavior (Excerpt)
private suspend fun testConnection(urlString: String, siteName: String): String {
    return withContext(Dispatchers.IO) {
        try {
            // Attempt connection to a site without SCT, such as "https://no-sct.
↪badssl.com"
            val url = URL(urlString)
            val connection = url.openConnection() as HttpURLConnection
            connection.connectTimeout = 8000
            connection.readTimeout = 8000
            connection.connect()
            val responseCode = connection.responseCode
            connection.disconnect()
            "Successfully connected to $siteName\n(HTTP Status: $responseCode)"
        } catch (e: SSLHandshakeException) {
            // In Android 17 (Target SDK 37), SSLHandshakeException is thrown here␣
↪because there is no SCT
            "Failed to connect to $siteName\nReason: SSLHandshakeException\n${e.
↪message}"
        } catch (e: Exception) {
            "Failed to connect to $siteName\nReason: ${e.javaClass.simpleName}: $
↪{e.message}"
        }
    }
}
```

Impact on Debugging and Staging Environments

In practice, caution is required when using self-signed certificates in staging environments, which are test environments prepared for final confirmation right before the production environment. Since a self-signed certificate is issued by oneself rather than a trusted CA, it is not registered in CT logs, which can cause communication errors in Android 17 environments. The same applies to development environments where certificates are replaced by internal proxies.

Customization using Network Security Configuration

For developers, it is important to check whether the certificates being used are CT-compliant, but public CAs such as DigiCert and Let's Encrypt are generally considered to have no issues. When using self-signed certificates in staging environments, it is recommended to explicitly set CT to false for the target domain in the Network Security Configuration file, or to use formal certificates.

```
<domain-config>
    <domain includeSubdomains="true">staging.example.com</domain>
    <certificateTransparency enabled="false"/>
</domain-config>
```

The benefits of implementing CT are detecting unauthorized certificates, deterring MITM attacks, and improving the transparency of CAs. Operational points to note are that internal and verification environments may stop working and that certificate management needs to be more rigorous. In Android 17, the world assumes that certificates are listed in CT logs, and because this feature affects all HTTPS communications and has a wide-ranging impact, developers need to be cautious.

5.4.3.11 Deprecation of usesCleartextTraffic and Permitting HTTP Communication

From Android 17, the usesCleartextTraffic attribute in AndroidManifest.xml, which has been used to permit HTTP communication, is deprecated, and its practical inactivation has begun in preparation for future removal.

Previously, configuring usesCleartextTraffic=”true” allowed HTTP communication uniformly for the entire application. However, starting with Android 17, even if this flag is turned on, the specification has been changed to block HTTP communication if a Network Security Configuration file does not exist.

In the future, when HTTP communication is required, rather than relying on permission from the flag alone, a security design is required to explicitly configure which domain to permit communication with using a Network Security Configuration file.

Permitting HTTP Communication with a Network Security Configuration File

When permitting HTTP communication to a specific domain (e.g., example.com), configure the Network Security Configuration file as follows. With this, HTTP communication to domains other than those explicitly permitted will be blocked.

```
<network-security-config>
    <base-config cleartextTrafficPermitted="false"/>
    <domain-config cleartextTrafficPermitted="true">
        <domain>example.com</domain>
    </domain-config>
</network-security-config>
```

Verification Example of OS Behavior Changes

The following verification code is an example of performing a GET request to a domain (http://example.com) for which HTTP communication is permitted by the Network Security Configuration file. In an Android 17 environment, if there is no permission configuration by a Network Security Configuration file, or if HTTP communication is performed to an unpermitted domain, the communication is blocked and an exception occurs.

```
// Verification code for performing HTTP communication (Excerpt)
suspend fun fetchData(urlString: String): String = withContext(Dispatchers.IO) {
    try {
        val url = URL(urlString)
        val connection = url.openConnection() as HttpURLConnection
        connection.requestMethod = "GET"
        connection.connectTimeout = 5000
        connection.readTimeout = 5000

        val responseCode = connection.responseCode
        if (responseCode == HttpURLConnection.HTTP_OK) {
            connection.inputStream.bufferedReader().use { it.readLine() ?: "Empty␣
↪response" }
        } else {
            "Error: $responseCode"
```

(continues on next page)

(continued from previous page)

```
        }
    } catch (e: Exception) {
        // If not permitted by the Network Security Configuration file,
        // an exception (such as Cleartext communication to [domain name] not␣
↪permitted by network security policy) is thrown here
        "Exception: ${e.message}"
    }
}
```

While a single flag previously permitted all HTTP communication, in Android 17 and later, communication is blocked unless the targets to be permitted are explicitly specified on a domain-by-domain basis. Therefore, developers must accurately grasp the external connection destinations with which the application communicates, and strictly manage the permissions, restricting them to the bare minimum.

5.5 Handling privacy data

In recent years, "Privacy-by-Design" concept has been proposed as a global trend to protect the privacy data. And based on the concept, governments are promoting legislation for privacy protection.

Applications that make use of user data in smartphones must take steps to ensure that users may use the application safely and securely without fears regarding privacy and personal data. These steps include handling user data appropriately and asking users to choose whether or not an application may use certain data. To this end, each application must prepare and display an application privacy policy indicating which information the application will use and how it will use that information; moreover, when fetching and using certain information, the application must first ask the user's permission. Note that application privacy policies differ from other documents that may have been present in the past—such as "Personal Data Protection Policies" or "Terms of Use"—and must be created separately from any such documents.

For details on the creation and execution of privacy policies, see the document "Smartphone Privacy Initiative" and "Smartphone Privacy Initiative II" (JMIC's SPI) released by Japan's Ministry of Internal Affairs and Communications (MIC).

The terminology used in this section is defined in the text and in Section "5.5.3.2. *Glossary of Terms*".

5.5.1 Sample Code

When preparing application privacy policy, you may use the "Tools to Assist in Creating Application Privacy Policies"[24]. These tools output two files—a summary version and a detailed version of the application privacy policy—both in HTML format and XML format. The HTML and XML content of these files comports with the recommendations of MIC's SPI including features such as search tags. In the sample code below, we will demonstrate the use of this tool to present application privacy policy using the HTML files prepared by this tool.

[24] http://www.kddi-research.jp/newsrelease/2013/090401.html

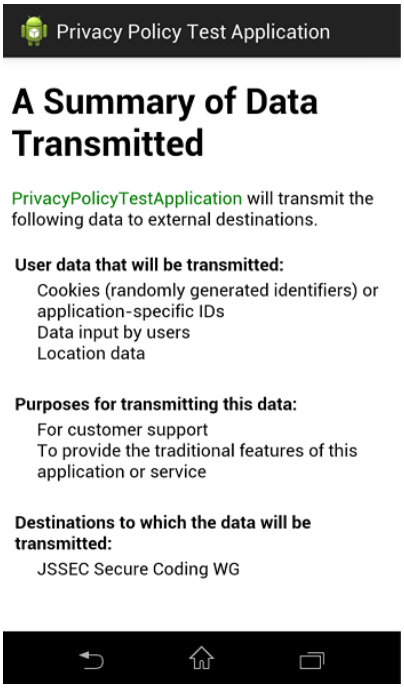


Fig. 5.5.1: Sample of Abstract Application Privacy Policy

More specifically, you may use the following flowchart to determine which sample code to use.

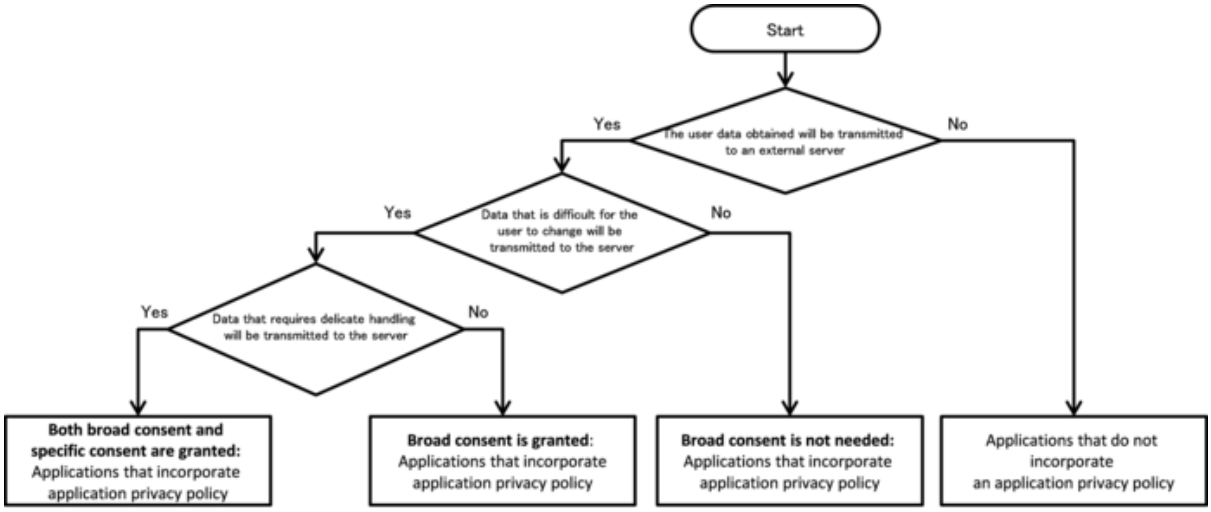


Fig. 5.5.2: Flow Figure to select sample code of handling privacy data

Here the phrase “broad consent” refers to a broad permission, granted by the user to the application upon the first launch of the application through display and review of the application privacy policy, for the application to transmit user data to servers.

In contrast, the phrase “specific consent” refers to pre consent obtained immediately prior to the transmission of specific user data.

5.5.1.1 Both broad consent and specific consent are granted: Applications that incorporate application privacy policy

Points: (Both broad consent and specific consent are granted: Applications that incorporate application privacy policy)

1. On first launch (or application update), obtain broad consent to transmit user data that will be handled by the application.
2. If the user does not grant broad consent, do not transmit user data.
3. **Obtain specific consent before transmitting user data that requires** particularly delicate handling.
4. **If the user does not grant specific consent, do not transmit the** corresponding data.
5. **Provide methods by which the user can review the application privacy** policy.
6. **Provide methods by which transmitted data can be deleted by user** operations.
7. **Provide methods by which transmitting data can be stopped by user** operations.
8. Use UUIDs or cookies to keep track of user data.
9. **Place a summary version of the application privacy policy in the** assets folder.

MainActivity.java

```
/*
 * Copyright (C) 2012-2026 Japan Smartphone Security Association
 *
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 *
 *      http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */

package org.jssec.android.privacypolicy;

import java.io.IOException;
import org.json.JSONException;
import org.json.JSONObject;
import org.jssec.android.privacypolicy.ConfirmFragment.DialogListener;

import com.google.android.gms.common.ConnectionResult;
import com.google.android.gms.common.GoogleApiAvailability;
import com.google.android.gms.location.FusedLocationProviderClient;
import com.google.android.gms.location.LocationServices;
import com.google.android.gms.tasks.OnSuccessListener;

import android.Manifest;
import android.location.Location;
```

(continues on next page)

(continued from previous page)

```
import android.os.AsyncTask;
import android.os.Build;
import android.os.Bundle;
import android.content.Intent;
import android.content.SharedPreferences;
import android.content.pm.PackageInfo;
import android.content.pm.PackageManager;
import android.content.pm.PackageManager.NameNotFoundException;
import androidx.core.app.ActivityCompat;
import androidx.fragment.app.FragmentActivity;
import androidx.fragment.app.FragmentManager;
import androidx.core.content.ContextCompat;
import android.text.Editable;
import android.text.TextWatcher;
import android.view.Menu;
import android.view.MenuItem;
import android.view.View;
import android.widget.TextView;
import android.widget.Toast;

public class MainActivity extends FragmentActivity implements  DialogListener {
    private static final String BASE_URL = "https://www.example.com/pp";
    private static final String GET_ID_URI = BASE_URL + "/get_id.php";
    private static final String SEND_DATA_URI = BASE_URL + "/send_data.php";
    private static final String DEL_ID_URI = BASE_URL + "/del_id.php";

    private static final String ID_KEY = "id";
    private static final String LOCATION_KEY = "location";
    private static final String NICK_NAME_KEY = "nickname";

    private static final String PRIVACY_POLICY_COMPREHENSIVE_AGREED_KEY =
            "privacyPolicyComprehensiveAgreed";
    private static final String PRIVACY_POLICY_DISCRETE_TYPE1_AGREED_KEY =
            "privacyPolicyDiscreteType1Agreed";
    private static final String PRIVACY_POLICY_PREF_NAME =
            "privacypolicy_preference";
    private static final int  MY_PERMISSIONS_REQUEST_ACCESS_FINE_LOCATION = 1;

    private String UserId = "";
    private FusedLocationProviderClient mFusedLocationClient;

    private final int DIALOG_TYPE_COMPREHENSIVE_AGREEMENT = 1;
    private final int DIALOG_TYPE_PRE_CONFIRMATION = 2;

    private static final int VERSION_TO_SHOW_COMPREHENSIVE_AGREEMENT_ANEW = 1;

    private TextWatcher watchHandler = new TextWatcher() {

        @Override
        public void beforeTextChanged(CharSequence s,
                                      int start, int count, int after) {
        }

        @Override
        public void onTextChanged(CharSequence s,
                                  int start, int before, int count) {
```

(continues on next page)

(continued from previous page)

```
            boolean buttonEnable = (s.length() > 0);

            MainActivity.this
                    .findViewById(R.id.buttonStart).setEnabled(buttonEnable);
        }

        @Override
        public void afterTextChanged(Editable s) {
        }
    };

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        mFusedLocationClient =
            LocationServices.getFusedLocationProviderClient(this);

        if (Build.VERSION.SDK_INT >= 23) {
            // API Level 23 or later requires permission for getting location info.
            int permissionCheck =
                ContextCompat.checkSelfPermission(this,
                    Manifest.permission.ACCESS_FINE_LOCATION);
            if (permissionCheck != PackageManager.PERMISSION_GRANTED) {
                // Because we have not permission, request it to user
                ActivityCompat.requestPermissions(this,
                    new String[]{Manifest.permission.ACCESS_FINE_LOCATION},
                    MY_PERMISSIONS_REQUEST_ACCESS_FINE_LOCATION);
            }
        }
    }

    @Override
    public void onRequestPermissionsResult(int requestCode,
                                           String permissions[],
                                           int[] grantResults) {
        switch (requestCode) {
            case MY_PERMISSIONS_REQUEST_ACCESS_FINE_LOCATION: {
                if (grantResults.length > 0
                        && grantResults[0] == PackageManager.PERMISSION_GRANTED) {
                    // Permission is granted
                    // Fetch user ID from server
                    new GetDataAsyncTask().execute();
                    findViewById(R.id.buttonStart).setEnabled(false);
                    ((TextView) findViewById(R.id.editTextNickname))
                        .addTextChangedListener(watchHandler);
                    int resultCode = GoogleApiAvailability
                                    .getInstance()
                                    .isGooglePlayServicesAvailable(this);
                    if (resultCode != ConnectionResult.SUCCESS) {
                        // Googleplay service is unavailable, our sample app will
                        // terminate.
                        finish();
                    }
                } else {
```

(continues on next page)

(continued from previous page)

```
                // Permission is not granted, sample app will terminate
                finish();
            }
        }
    }
}

@Override
protected void onStart() {
    super.onStart();

    SharedPreferences pref =
        getSharedPreferences(PRIVACY_POLICY_PREF_NAME, MODE_PRIVATE);
    int privacyPolicyAgreed =
        pref.getInt(PRIVACY_POLICY_COMPREHENSIVE_AGREED_KEY, -1);
    if (privacyPolicyAgreed <= VERSION_TO_SHOW_COMPREHENSIVE_AGREEMENT_ANEW) {
        // *** POINT 1 *** On first launch (or application update),
        // obtain broad consent to transmit user data that will be handled
        // by the application.
        // When the application is updated, it is only necessary to renew
        // the user's grant of broad consent
        // if the updated application will handle new types of user data.
        ConfirmFragment dialog =
            ConfirmFragment.newInstance(R.string.privacyPolicy,
                                        R.string.agreePrivacyPolicy,
                                        DIALOG_TYPE_COMPREHENSIVE_AGREEMENT);
        dialog.setDialogListener(this);
        FragmentManager fragmentManager = getSupportFragmentManager();
        dialog.show(fragmentManager, "dialog");
    }
}

public void onSendToServer(View view) {
    // Check the status of user consent.
    // Actually, it is necessary to obtain consent for each user data type.
    SharedPreferences pref =
            getSharedPreferences(PRIVACY_POLICY_PREF_NAME, MODE_PRIVATE);
    int privacyPolicyAgreed =
            pref.getInt(PRIVACY_POLICY_DISCRETE_TYPE1_AGREED_KEY, -1);
    if (privacyPolicyAgreed <= VERSION_TO_SHOW_COMPREHENSIVE_AGREEMENT_ANEW) {
        // *** POINT 3 *** Obtain specific consent before transmitting user
        // data that requires particularly delicate handling.
        ConfirmFragment dialog =
            ConfirmFragment.newInstance(R.string.sendLocation,
                                        R.string.cofirmSendLocation,
                                        DIALOG_TYPE_PRE_CONFIRMATION);
        dialog.setDialogListener(this);
        FragmentManager fragmentManager = getSupportFragmentManager();
        dialog.show(fragmentManager, "dialog");
    } else {
        // Start transmission, since it has the user consent.
        onPositiveButtonClick(DIALOG_TYPE_PRE_CONFIRMATION);
    }
}

public void onPositiveButtonClick(int type) {
```

(continues on next page)

(continued from previous page)

```
        if (type == DIALOG_TYPE_COMPREHENSIVE_AGREEMENT) {
            // *** POINT 1 *** On first launch (or application update),
            // obtain broad consent to transmit user data that will be handled by
            // the application.
            SharedPreferences.Editor pref = getSharedPreferences(PRIVACY_POLICY_
↪PREF_NAME, MODE_PRIVATE).edit();
            pref.putInt(PRIVACY_POLICY_COMPREHENSIVE_AGREED_KEY, getVersionCode());
            pref.apply();
        } else if (type == DIALOG_TYPE_PRE_CONFIRMATION) {
            // *** POINT 3 *** Obtain specific consent before transmitting user
            // data that requires particularly delicate handling.
            mFusedLocationClient.getLastLocation()
                .addOnSuccessListener(this, new OnSuccessListener<Location>() {
                @Override
                public void onSuccess(Location location) {
                    String nickname =
                        ((TextView) findViewById(R.id.editTextNickname))
                            .getText().toString();
                    if (location != null) {
                        String locationData =
                            "Latitude:" + location.getLatitude() +
                            ", Longitude:" + location.getLongitude();
                        Toast.makeText(MainActivity.this,
                            this.getClass().getSimpleName() +
                            "\n - nickname : " + nickname +
                            "\n - location : " + locationData,
                            Toast.LENGTH_SHORT).show();
                        new SendDataAsyncTack().execute(SEND_DATA_URI,
                            UserId, locationData, nickname);
                    } else {
                        Toast.makeText(MainActivity.this,
                            this.getClass().getSimpleName() +
                            "\n - nickname : " + nickname +
                            "\n - location : unavailable",
                            Toast.LENGTH_SHORT).show();
                    }
                }
            });
            // Store the status of user consent.
            // Actually, it is necessary to obtain consent for each user data type.
            SharedPreferences.Editor pref =
                getSharedPreferences(PRIVACY_POLICY_PREF_NAME, MODE_PRIVATE)
                .edit();
            pref.putInt(PRIVACY_POLICY_DISCRETE_TYPE1_AGREED_KEY,
                        getVersionCode());
            pref.apply();
        }
    }

    public void onNegativeButtonClick(int type) {
        if (type == DIALOG_TYPE_COMPREHENSIVE_AGREEMENT) {
            // *** POINT 2 *** If the user does not grant general consent, do not
            // transmit user data.
            // In this sample application we terminate the application in this
            // case.
            finish();
```

(continues on next page)

(continued from previous page)

```
        } else if (type == DIALOG_TYPE_PRE_CONFIRMATION) {
            // *** POINT 4 *** If the user does not grant specific consent, do not
            // transmit the corresponding data.
            // The user did not grant consent, so we do nothing.
        }
    }

    private int getVersionCode() {
        int versionCode = -1;
        PackageManager packageManager = this.getPackageManager();
        try {
            PackageInfo packageInfo =
                packageManager.getPackageInfo(this.getPackageName(),
                                              PackageManager.GET_ACTIVITIES);
            versionCode = packageInfo.versionCode;
        } catch (NameNotFoundException e) {
            // This is sample, so omit the exception process
        }

        return versionCode;
    }

    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        getMenuInflater().inflate(R.menu.main, menu);
        return true;
    }

    @Override
    public boolean onOptionsItemSelected(MenuItem item) {
        int itemId = item.getItemId();
        if (itemId == R.id.action_show_pp) {
            // *** POINT 5 *** Provide methods by which the user can review the
            // application privacy policy.
            Intent intent = new Intent();
            intent.setClass(this, WebViewAssetsActivity.class);
            startActivity(intent);
            return true;
        } else if (itemId == R.id.action_del_id) {
            // *** POINT 6 *** Provide methods by which transmitted data can be
            // deleted by user operations.
            new SendDataAsyncTack().execute(DEL_ID_URI, UserId);
            return true;
        } else if (itemId == R.id.action_donot_send_id) {
            // *** POINT 7 *** Provide methods by which transmitting data can be
            // stopped by user operations.

            // If the user stop sending data, user consent is deemed to have been
            // revoked.
            SharedPreferences.Editor pref = getSharedPreferences(PRIVACY_POLICY_
↪PREF_NAME, MODE_PRIVATE).edit();
            pref.putInt(PRIVACY_POLICY_COMPREHENSIVE_AGREED_KEY, 0);
            pref.apply();

            // In this sample application if the user data cannot be sent by user
            // operations, finish the application because we do nothing.
```

(continues on next page)

(continued from previous page)

```
            String message  = getString(R.string.stopSendUserData);
            Toast.makeText(MainActivity.this,
                this.getClass().getSimpleName() + " - " + message,
                Toast.LENGTH_SHORT).show();
            finish();

            return true;
        }

        return false;
    }

    private class GetDataAsyncTask extends AsyncTask<String, Void, String> {
        private String extMessage = "";

        @Override
        protected String doInBackground(String... params) {
            // *** POINT 8 *** Use UUIDs or cookies to keep track of user data
            // In this sample we use an ID generated on the server side
            SharedPreferences sp =
                getSharedPreferences(PRIVACY_POLICY_PREF_NAME, MODE_PRIVATE);
            UserId = sp.getString(ID_KEY, null);
            if (UserId == null) {
                // There is not token in SharedPreferences, obtain ID from server
                try {
                    UserId = NetworkUtil.getCookie(GET_ID_URI, "", "id");
                } catch (IOException e) {
                    // Handle exception such as certificate error
                    extMessage = e.toString();
                }
                // Save obtained ID in SharedPreferences
                sp.edit().putString(ID_KEY, UserId).commit();
            }
            return UserId;
        }

        @Override
        protected void onPostExecute(final String data) {
            String status = (data != null) ? "success" : "error";
            Toast.makeText(MainActivity.this,
                           this.getClass().getSimpleName() +
                           " - " + status + " : " + extMessage,
                           Toast.LENGTH_SHORT).show();
        }
    }

    private class SendDataAsyncTack extends AsyncTask<String, Void, Boolean> {
        private String extMessage = "";

        @Override
        protected Boolean doInBackground(String... params) {
            String url = params[0];
            String id = params[1];
            String location = params.length > 2 ? params[2] : null;
            String nickname = params.length > 3 ? params[3] : null;
```

(continues on next page)

(continued from previous page)

```
            Boolean result = false;
            try {
                JSONObject jsonData = new JSONObject();
                jsonData.put(ID_KEY, id);
                if (location != null)
                    jsonData.put(LOCATION_KEY, location);

                if (nickname != null)
                    jsonData.put(NICK_NAME_KEY, nickname);

                NetworkUtil.sendJSON(url, "", jsonData.toString());

                result = true;
            } catch (IOException e) {
                // Catch exceptions such as certification errors
                extMessage = e.toString();
            } catch (JSONException e) {
                extMessage = e.toString();
            }
            return result;
        }

        @Override
        protected void onPostExecute(Boolean result) {
            String status = result ? "Success" : "Error";
            Toast.makeText(MainActivity.this,
                           this.getClass().getSimpleName() +
                           " - " + status + " : " + extMessage,
                           Toast.LENGTH_SHORT).show();
        }
    }
}
```

ConfirmFragment.java

```
/*
 * Copyright (C) 2012-2026 Japan Smartphone Security Association
 *
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 *
 *      http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */

package org.jssec.android.privacypolicy;

import android.app.Activity;
import android.app.AlertDialog;
import android.app.Dialog;
import android.content.Context;
```

(continues on next page)

(continued from previous page)

```
import android.content.DialogInterface;
import android.content.Intent;
import android.os.Bundle;
import androidx.fragment.app.DialogFragment;
import android.view.LayoutInflater;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.TextView;

public class ConfirmFragment extends DialogFragment {

    private DialogListener mListener = null;

    public static interface DialogListener {
        public void onPositiveButtonClick(int type);

        public void onNegativeButtonClick(int type);
    }

    public static ConfirmFragment newInstance(int title, int sentence, int type) {
        ConfirmFragment fragment = new ConfirmFragment();
        Bundle args = new Bundle();
        args.putInt("title", title);
        args.putInt("sentence", sentence);
        args.putInt("type", type);
        fragment.setArguments(args);
        return fragment;
    }

    @Override
    public Dialog onCreateDialog(Bundle args) {
        // *** POINT 1 *** On first launch (or application update), obtain broad
        // consent to transmit user data that will be handled by the application.
        // *** POINT 3 *** Obtain specific consent before transmitting user data
        // that requires particularly delicate handling.
        final int title = getArguments().getInt("title");
        final int sentence = getArguments().getInt("sentence");
        final int type = getArguments().getInt("type");

        LayoutInflater inflater = (LayoutInflater) getActivity()
                .getSystemService(Context.LAYOUT_INFLATER_SERVICE);
        View content = inflater.inflate(R.layout.fragment_comfirm, null);
        TextView linkPP = (TextView) content.findViewById(R.id.tx_link_pp);
        linkPP.setOnClickListener(new OnClickListener() {
            @Override
            public void onClick(View v) {
                // *** POINT 5 *** Provide methods by which the user can review
                // the application privacy policy.
                Intent intent = new Intent();
                intent.setClass(getActivity(), WebViewAssetsActivity.class);
                startActivity(intent);
            }
        });

        AlertDialog.Builder builder = new AlertDialog.Builder(getActivity());
        builder.setIcon(R.drawable.ic_launcher);
```

(continues on next page)

(continued from previous page)

```
        builder.setTitle(title);
        builder.setMessage(sentence);
        builder.setView(content);

        builder.setPositiveButton(R.string.buttonConsent,
            new DialogInterface.OnClickListener() {
            public void onClick(DialogInterface dialog, int whichButton) {
                if (mListener != null) {
                    mListener.onPositiveButtonClick(type);
                }
            }
        });
        builder.setNegativeButton(R.string.buttonDonotConsent,
            new DialogInterface.OnClickListener() {
            public void onClick(DialogInterface dialog, int whichButton) {
                if (mListener != null) {
                    mListener.onNegativeButtonClick(type);
                }
            }
        });

        Dialog dialog = builder.create();
        dialog.setCanceledOnTouchOutside(false);

        return dialog;
    }

    @Override
    public void onAttach(Activity activity) {
        super.onAttach(activity);
        if (!(activity instanceof DialogListener)) {
            throw new ClassCastException(activity.toString() +
                    " must implement DialogListener.");
        }
        mListener = (DialogListener) activity;
    }

    public void setDialogListener(DialogListener listener) {
        mListener = listener;
    }
}
```

WebViewAssetsActivity.java

```
/*
 * Copyright (C) 2012-2026 Japan Smartphone Security Association
 *
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 *
 *      http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
```

(continues on next page)

(continued from previous page)

```
 * limitations under the License.
 */

package org.jssec.android.privacypolicy;

import android.app.Activity;
import android.os.Bundle;
import android.webkit.WebSettings;
import android.webkit.WebView;

public class WebViewAssetsActivity extends Activity {
    // *** POINT 9 *** Place a summary version of the application privacy policy
    // in the assets folder
    private static final String ABST_PP_URL =
        "file:///android_asset/PrivacyPolicy/app-policy-abst-privacypolicy-1.0.html
↪";

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_webview);

        WebView webView = (WebView) findViewById(R.id.webView);
        WebSettings webSettings = webView.getSettings();

        webSettings.setAllowFileAccess(false);

        webView.loadUrl(ABST_PP_URL);
    }
}
```

5.5.1.2 Broad consent is granted: Applications that incorporate application privacy policy

Points: (Broad consent is granted: Applications that incorporate application privacy policy)

1. On first launch (or application update), obtain broad consent to transmit user data that will be handled by the application.
2. If the user does not grant broad consent, do not transmit user data.
3. **Provide methods by which the user can review the application privacy policy.**
4. **Provide methods by which transmitted data can be deleted by user operations.**
5. **Provide methods by which transmitting data can be stopped by user operations.**
6. Use UUIDs or cookies to keep track of user data.
7. **Place a summary version of the application privacy policy in the assets folder.**

MainActivity.java

```
/*
 * Copyright (C) 2012-2026 Japan Smartphone Security Association
 *
 * Licensed under the Apache License, Version 2.0 (the "License");
```

(continues on next page)

(continued from previous page)

```
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 *
 *      http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */

package org.jssec.android.privacypolicynopreconfirm;

import java.io.IOException;
import java.util.UUID;

import org.json.JSONException;
import org.json.JSONObject;
import org.jssec.android.privacypolicynopreconfirm.ConfirmFragment.DialogListener;

import android.os.AsyncTask;
import android.os.Bundle;
import android.content.Intent;
import android.content.SharedPreferences;
import android.content.pm.PackageInfo;
import android.content.pm.PackageManager;
import android.content.pm.PackageManager.NameNotFoundException;

import androidx.fragment.app.FragmentActivity;
import androidx.fragment.app.FragmentManager;
import android.text.Editable;
import android.text.TextWatcher;
import android.view.Menu;
import android.view.MenuItem;
import android.view.View;
import android.widget.TextView;
import android.widget.Toast;

public class MainActivity extends FragmentActivity implements DialogListener {
    private final String BASE_URL = "https://www.example.com/pp";
    private final String GET_ID_URI = BASE_URL + "/get_id.php";
    private final String SEND_DATA_URI = BASE_URL + "/send_data.php";
    private final String DEL_ID_URI = BASE_URL + "/del_id.php";

    private final String ID_KEY = "id";
    private final String NICK_NAME_KEY = "nickname";
    private final String LN_KEY = "lineNumber";

    private final String PRIVACY_POLICY_AGREED_KEY = "privacyPolicyAgreed";

    private final String PRIVACY_POLICY_PREF_NAME = "privacypolicy_preference";

    private String mUUId = "";

    private String UserId = "";
```

(continues on next page)

(continued from previous page)

```
    private final int DIALOG_TYPE_COMPREHENSIVE_AGREEMENT = 1;

    private final int VERSION_TO_SHOW_COMPREHENSIVE_AGREEMENT_ANEW = 1;

    private TextWatcher watchHandler = new TextWatcher() {

        @Override
        public void beforeTextChanged(CharSequence s,
                                      int start, int count, int after) {
        }

        @Override
        public void onTextChanged(CharSequence s,
                                  int start, int before, int count) {
            boolean buttonEnable = (s.length() > 0);

            MainActivity.this.findViewById(R.id.buttonStart)
                             .setEnabled(buttonEnable);
        }

        @Override
        public void afterTextChanged(Editable s) {
        }
    };

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        // send randomly generated UUID
        mUUId = UUID.randomUUID().toString();

        // Fetch user ID from serverFetch user ID from server
        new GetDataAsyncTask().execute();

        findViewById(R.id.buttonStart).setEnabled(false);
        ((TextView) findViewById(R.id.editTextNickname))
            .addTextChangedListener(watchHandler);
    }

    @Override
    protected void onStart() {
        super.onStart();

        SharedPreferences pref =
                getSharedPreferences(PRIVACY_POLICY_PREF_NAME, MODE_PRIVATE);
        int privacyPolicyAgreed =
                pref.getInt(PRIVACY_POLICY_AGREED_KEY, -1);

        if (privacyPolicyAgreed <= VERSION_TO_SHOW_COMPREHENSIVE_AGREEMENT_ANEW) {
            // *** POINT 1 *** On first launch (or application update), obtain
            // broad consent to transmit user data that will be handled by the
            // application.
            // When the application is updated, it is only necessary to renew
```

(continues on next page)

(continued from previous page)

```
            // the user's grant of broad consent if the updated application
            // will handle new types of user data.
            ConfirmFragment dialog =
                ConfirmFragment.newInstance(R.string.privacyPolicy,
                                            R.string.agreePrivacyPolicy,
                                            DIALOG_TYPE_COMPREHENSIVE_AGREEMENT);
            dialog.setDialogListener(this);
            FragmentManager fragmentManager = getSupportFragmentManager();
            dialog.show(fragmentManager, "dialog");
        }
    }

    public void onSendToServer(View view) {
        String nickname =
            ((TextView) findViewById(R.id.editTextNickname)).getText().toString();
        Toast.makeText(MainActivity.this,
                  this.getClass().getSimpleName()
                       + "\n - nickname : " + nickname + ", UUID = " + mUUId,
            Toast.LENGTH_SHORT).show();
        new SendDataAsyncTack().execute(SEND_DATA_URI, UserId, nickname, mUUId);
    }

    public void onPositiveButtonClick(int type) {
        if (type == DIALOG_TYPE_COMPREHENSIVE_AGREEMENT) {
            // *** POINT 1 *** On first launch (or application update), obtain
            // broad consent to transmit user data that will be handled by the
            // application.
            SharedPreferences.Editor pref =
                getSharedPreferences(PRIVACY_POLICY_PREF_NAME, MODE_PRIVATE)
                .edit();
            pref.putInt(PRIVACY_POLICY_AGREED_KEY, getVersionCode());
            pref.apply();
        }
    }

    public void onNegativeButtonClick(int type) {
        if (type == DIALOG_TYPE_COMPREHENSIVE_AGREEMENT) {
            // *** POINT 2 *** If the user does not grant general consent, do not
            // transmit user data.
            // In this sample application we terminate the application in this
            // case.
            finish();
        }
    }

    private int getVersionCode() {
        int versionCode = -1;
        PackageManager packageManager = this.getPackageManager();
        try {
            PackageInfo packageInfo =
                packageManager.getPackageInfo(this.getPackageName(),
                                              PackageManager.GET_ACTIVITIES);
            versionCode = packageInfo.versionCode;
        } catch (NameNotFoundException e) {
            // This is sample, so omit the exception process
        }
```

(continues on next page)

(continued from previous page)

```
        return versionCode;
    }

    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        getMenuInflater().inflate(R.menu.main, menu);
        return true;
    }

    @Override
    public boolean onOptionsItemSelected(MenuItem item) {
        int itemId = item.getItemId();
        if (itemId == R.id.action_show_pp) {
            // *** POINT 3 *** Provide methods by which the user can review the
            // application privacy policy.
            Intent intent = new Intent();
            intent.setClass(this, WebViewAssetsActivity.class);
            startActivity(intent);
            return true;
        } else if (itemId == R.id.action_del_id) {
            // *** POINT 4 *** Provide methods by which transmitted data can be
            // deleted by user operations.
            new SendDataAsyncTack().execute(DEL_ID_URI, UserId);
            return true;
        } else if (itemId == R.id.action_donot_send_id) {
            // *** POINT 5 *** Provide methods by which transmitting data can be
            // stopped by user operations.

            // If the user stop sending data, user consent is deemed to have been
            // revoked.
            SharedPreferences.Editor pref =
                getSharedPreferences(PRIVACY_POLICY_PREF_NAME, MODE_PRIVATE)
                .edit();
            pref.putInt(PRIVACY_POLICY_AGREED_KEY, 0);
            pref.apply();

            // In this sample application if the user data cannot be sent by user
            // operations, finish the application because we do nothing.
            String message  = getString(R.string.stopSendUserData);
            Toast.makeText(MainActivity.this,
                this.getClass().getSimpleName() + " - " + message,
                Toast.LENGTH_SHORT).show();
            finish();

            return true;                }
        return false;
    }

    private class GetDataAsyncTask extends AsyncTask<String, Void, String> {
        private String extMessage = "";

        @Override
        protected String doInBackground(String... params) {
            // *** POINT 6 *** Use UUIDs or cookies to keep track of user data
            // In this sample we use an ID generated on the server side
```

(continues on next page)

(continued from previous page)

```
            SharedPreferences sp = getSharedPreferences(PRIVACY_POLICY_PREF_NAME,
                                                        MODE_PRIVATE);
            UserId = sp.getString(ID_KEY, null);
            if (UserId == null) {
                // No token in SharedPreferences; fetch ID from server
                try {
                    UserId = NetworkUtil.getCookie(GET_ID_URI, "", "id");
                } catch (IOException e) {
                    // Catch exceptions such as certification errors
                    extMessage = e.toString();
                }

                // Store the fetched ID in SharedPreferences
                sp.edit().putString(ID_KEY, UserId).commit();
            }
            return UserId;
        }

        @Override
        protected void onPostExecute(final String data) {
            String status = (data != null) ? "success" : "error";
            Toast.makeText(MainActivity.this,
                           this.getClass().getSimpleName() +
                           " - " + status + " : " + extMessage,
                           Toast.LENGTH_SHORT).show();
        }
    }

    private class SendDataAsyncTack extends AsyncTask<String, Void, Boolean> {
        private String extMessage = "";

        @Override
        protected Boolean doInBackground(String... params) {
            String url = params[0];
            String id = params[1];
            String nickname = params.length > 2 ? params[2] : null;
            String lineNum = params.length > 3 ? params[3] : null;

            Boolean result = false;
            try {
                JSONObject jsonData = new JSONObject();
                jsonData.put(ID_KEY, id);

                if (nickname != null)
                    jsonData.put(NICK_NAME_KEY, nickname);

                if (lineNum != null)
                    jsonData.put(LN_KEY, lineNum);

                NetworkUtil.sendJSON(url, "", jsonData.toString());

                result = true;
            } catch (IOException e) {
                // Catch exceptions such as certification errors
                extMessage = e.toString();
            } catch (JSONException e) {
```

(continues on next page)

(continued from previous page)

```
                extMessage = e.toString();
            }
            return result;
        }

        @Override
        protected void onPostExecute(Boolean result) {
            String status = result ? "Success" : "Error";
            Toast.makeText(MainActivity.this,
                           this.getClass().getSimpleName() +
                           " - " + status + " : " + extMessage,
                           Toast.LENGTH_SHORT).show();
        }
    }
}
```

ConfirmFragment.java

```
/*
 * Copyright (C) 2012-2026 Japan Smartphone Security Association
 *
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 *
 *      http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */

package org.jssec.android.privacypolicynopreconfirm;

import android.app.Activity;
import android.app.AlertDialog;
import android.app.Dialog;
import android.content.Context;
import android.content.DialogInterface;
import android.content.Intent;
import android.os.Bundle;
import androidx.fragment.app.DialogFragment;
import android.view.LayoutInflater;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.TextView;

public class ConfirmFragment extends DialogFragment {

    private DialogListener mListener = null;

    public static interface DialogListener {
        public void onPositiveButtonClick(int type);

        public void onNegativeButtonClick(int type);
```

(continues on next page)

(continued from previous page)

```
    }

    public static ConfirmFragment newInstance(int title, int sentence, int type) {
        ConfirmFragment fragment = new ConfirmFragment();
        Bundle args = new Bundle();
        args.putInt("title", title);
        args.putInt("sentence", sentence);
        args.putInt("type", type);
        fragment.setArguments(args);
        return fragment;
    }

    @Override
    public Dialog onCreateDialog(Bundle args) {
        // *** POINT 1 *** On first launch (or application update), obtain broad
        // consent to transmit user data that will be handled by the application.
        final int title = getArguments().getInt("title");
        final int sentence = getArguments().getInt("sentence");
        final int type = getArguments().getInt("type");

        LayoutInflater inflater = (LayoutInflater) getActivity()
                .getSystemService(Context.LAYOUT_INFLATER_SERVICE);
        View content = inflater.inflate(R.layout.fragment_comfirm, null);
        TextView linkPP = (TextView) content.findViewById(R.id.tx_link_pp);
        linkPP.setOnClickListener(new OnClickListener() {
            @Override
            public void onClick(View v) {
                // *** POINT 3 *** Provide methods by which the user can review
                // the application privacy policy.
                Intent intent = new Intent();
                intent.setClass(getActivity(), WebViewAssetsActivity.class);
                startActivity(intent);
            }
        });

        AlertDialog.Builder builder = new AlertDialog.Builder(getActivity());
        builder.setIcon(R.drawable.ic_launcher);
        builder.setTitle(title);
        builder.setMessage(sentence);
        builder.setView(content);

        builder.setPositiveButton(R.string.buttonConsent,
                new DialogInterface.OnClickListener() {
                public void onClick(DialogInterface dialog, int whichButton) {
                    if (mListener != null) {
                        mListener.onPositiveButtonClick(type);
                    }
                }
            });
        builder.setNegativeButton(R.string.buttonDonotConsent,
                new DialogInterface.OnClickListener() {
                public void onClick(DialogInterface dialog, int whichButton) {
                    if (mListener != null) {
                        mListener.onNegativeButtonClick(type);
                    }
                }
```

(continues on next page)

(continued from previous page)

```
            });

        Dialog dialog = builder.create();
        dialog.setCanceledOnTouchOutside(false);

        return dialog;
    }

    @Override
    public void onAttach(Activity activity) {
        super.onAttach(activity);
        if (!(activity instanceof DialogListener)) {
            throw new ClassCastException(activity.toString()
                                         + " must implement DialogListener.");
        }
        mListener = (DialogListener) activity;
    }

    public void setDialogListener(DialogListener listener) {
        mListener = listener;
    }
}
```

WebViewAssetsActivity.java

```
/*
 * Copyright (C) 2012-2026 Japan Smartphone Security Association
 *
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 *
 *      http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */

package org.jssec.android.privacypolicynopreconfirm;

import org.jssec.android.privacypolicynopreconfirm.R;

import android.app.Activity;
import android.os.Bundle;
import android.webkit.WebSettings;
import android.webkit.WebView;

public class WebViewAssetsActivity extends Activity {
    // *** POINT 7 *** Place a summary version of the application privacy policy
    // in the assets folder
    private final String ABST_PP_URL =
        "file:///android_asset/PrivacyPolicy/app-policy-abst-privacypolicy-1.0.html
↪";
```

(continues on next page)

(continued from previous page)

```
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_webview);

        WebView webView = (WebView) findViewById(R.id.webView);
        WebSettings webSettings = webView.getSettings();

        webSettings.setAllowFileAccess(false);

        webView.loadUrl(ABST_PP_URL);
    }
}
```

5.5.1.3 Broad consent is not needed: Applications that incorporate application privacy policy

Points: (Broad consent is not needed: Applications that incorporate application privacy policy)

1. Provide methods by which the user can review the application privacy policy.
2. Provide methods by which transmitted data can be deleted by user operations.
3. Provide methods by which transmitting data can be stopped by user operations
4. Use UUIDs or cookies to keep track of user data.
5. Place a summary version of the application privacy policy in the assets folder.

MainActivity.java

```
/*
 * Copyright (C) 2012-2026 Japan Smartphone Security Association
 *
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 *
 *      http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */

package org.jssec.android.privacypolicynocomprehensive;

import java.io.IOException;
import org.json.JSONException;
import org.json.JSONObject;

import android.os.AsyncTask;
import android.os.Bundle;
import android.content.Intent;
import android.content.SharedPreferences;
import androidx.fragment.app.FragmentActivity;
import android.text.Editable;
import android.text.TextWatcher;
```

(continues on next page)

(continued from previous page)

```
import android.view.Menu;
import android.view.MenuItem;
import android.view.View;
import android.widget.TextView;
import android.widget.Toast;

public class MainActivity extends FragmentActivity {
    private static final String BASE_URL = "https://www.example.com/pp";
    private static final String GET_ID_URI = BASE_URL + "/get_id.php";
    private static final String SEND_DATA_URI = BASE_URL + "/send_data.php";
    private static final String DEL_ID_URI = BASE_URL + "/del_id.php";

    private static final String ID_KEY = "id";
    private static final String NICK_NAME_KEY = "nickname";

    private static final String PRIVACY_POLICY_PREF_NAME =
            "privacypolicy_preference";

    private String UserId = "";

    private TextWatcher watchHandler = new TextWatcher() {

        @Override
        public void beforeTextChanged(CharSequence s,
                                      int start, int count, int after) {
        }

        @Override
        public void onTextChanged(CharSequence s,
                                  int start, int before, int count) {
            boolean buttonEnable = (s.length() > 0);

            MainActivity.this.findViewById(R.id.buttonStart)
                             .setEnabled(buttonEnable);
        }

        @Override
        public void afterTextChanged(Editable s) {
        }
    };

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        // Fetch user ID from serverFetch user ID from server
        new GetDataAsyncTask().execute();

        findViewById(R.id.buttonStart).setEnabled(false);
        ((TextView) findViewById(R.id.editTextNickname))
                .addTextChangedListener(watchHandler);
    }

    public void onSendToServer(View view) {
        String nickname =
```

(continues on next page)

(continued from previous page)

```
            ((TextView) findViewById(R.id.editTextNickname))
                    .getText().toString();
        Toast.makeText(MainActivity.this,
                       this.getClass().getSimpleName() +
                       "\n - nickname : " + nickname,
                       Toast.LENGTH_SHORT).show();
        new sendDataAsyncTack().execute(SEND_DATA_URI, UserId, nickname);
    }

    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        getMenuInflater().inflate(R.menu.main, menu);
        return true;
    }

    @Override
    public boolean onOptionsItemSelected(MenuItem item) {
        int itemId = item.getItemId();
        if (itemId == R.id.action_show_pp) {
            // *** POINT 1 *** Provide methods by which the user can review the
            // application privacy policy.
            Intent intent = new Intent();
            intent.setClass(this, WebViewAssetsActivity.class);
            startActivity(intent);
            return true;
        } else if (itemId == R.id.action_del_id) {
            // *** POINT 2 *** Provide methods by which transmitted data can be
            // deleted by user operations.
            new sendDataAsyncTack().execute(DEL_ID_URI, UserId);
            return true;
        } else if (itemId == R.id.action_donot_send_id) {
            // *** POINT 3 *** Provide methods by which transmitting data can be
            // stopped by user operations.

            // In this sample application if the user data cannot be sent by user
            // operations, finish the application because we do nothing.
            String message  = getString(R.string.stopSendUserData);
            Toast.makeText(MainActivity.this,
                this.getClass().getSimpleName() + " - " + message,
                Toast.LENGTH_SHORT).show();
            finish();

            return true;
        }
        return false;
    }

    private class GetDataAsyncTask extends AsyncTask<String, Void, String> {
        private String extMessage = "";

        @Override
        protected String doInBackground(String... params) {
            // *** POINT 4 *** Use UUIDs or cookies to keep track of user data
            // In this sample we use an ID generated on the server side
            SharedPreferences sp =
                getSharedPreferences(PRIVACY_POLICY_PREF_NAME, MODE_PRIVATE);
```

(continues on next page)

(continued from previous page)

```
            UserId = sp.getString(ID_KEY, null);
            if (UserId == null) {
                // No token in SharedPreferences; fetch ID from server
                try {
                    UserId = NetworkUtil.getCookie(GET_ID_URI, "", "id");
                } catch (IOException e) {
                    // Catch exceptions such as certification errors
                    extMessage = e.toString();
                }

                // Store the fetched ID in SharedPreferences
                sp.edit().putString(ID_KEY, UserId).commit();
            }
            return UserId;
        }

        @Override
        protected void onPostExecute(final String data) {
            String status = (data != null) ? "success" : "error";
            Toast.makeText(MainActivity.this,
                           this.getClass().getSimpleName() +
                           " - " + status + " : " + extMessage,
                           Toast.LENGTH_SHORT).show();
        }
    }

    private class sendDataAsyncTack extends AsyncTask<String, Void, Boolean> {
        private String extMessage = "";

        @Override
        protected Boolean doInBackground(String... params) {
            String url = params[0];
            String id = params[1];
            String nickname = params.length > 2 ? params[2] : null;

            Boolean result = false;
            try {
                JSONObject jsonData = new JSONObject();
                jsonData.put(ID_KEY, id);

                if (nickname != null)
                    jsonData.put(NICK_NAME_KEY, nickname);

                NetworkUtil.sendJSON(url, "", jsonData.toString());

                result = true;
            } catch (IOException e) {
                // Catch exceptions such as certification errors
                extMessage = e.toString();
            } catch (JSONException e) {
                extMessage = e.toString();
            }
            return result;
        }

        @Override
```

(continues on next page)

(continued from previous page)

```
        protected void onPostExecute(Boolean result) {
            String status = result ? "Success" : "Error";
            Toast.makeText(MainActivity.this,
                           this.getClass().getSimpleName() +
                           " - " + status + " : " + extMessage,
                           Toast.LENGTH_SHORT).show();
        }
    }
}
```

WebViewAssetsActivity.java

```
/*
 * Copyright (C) 2012-2026 Japan Smartphone Security Association
 *
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 *
 *      http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */

package org.jssec.android.privacypolicynocomprehensive;

import org.jssec.android.privacypolicynocomprehensive.R;

import android.app.Activity;
import android.os.Bundle;
import android.webkit.WebSettings;
import android.webkit.WebView;

public class WebViewAssetsActivity extends Activity {
    // *** POINT 5 *** Place a summary version of the application privacy policy
    // in the assets folder
    private static final String ABST_PP_URL =
        "file:///android_asset/PrivacyPolicy/app-policy-abst-privacypolicy-1.0.html
↪";

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_webview);

        WebView webView = (WebView) findViewById(R.id.webView);
        WebSettings webSettings = webView.getSettings();

        webSettings.setAllowFileAccess(false);

        webView.loadUrl(ABST_PP_URL);
    }
}
```

5.5.1.4 Applications that do not incorporate an application privacy policy

Points: (Applications that do not incorporate an application privacy policy)

1. You do not need to display an application privacy policy if your application will only use the information it obtains within the device.
2. **In the documentation for marketplace applications or similar** applications, note that the application does not transmit the information it obtains to the outside world

MainActivity.java

```
/*
 * Copyright (C) 2012-2026 Japan Smartphone Security Association
 *
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 *
 *      http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */

package org.jssec.android.privacypolicynoinfosent;

import com.google.android.gms.common.ConnectionResult;
import com.google.android.gms.common.GoogleApiAvailability;
import com.google.android.gms.location.FusedLocationProviderClient;
import com.google.android.gms.location.LocationServices;
import com.google.android.gms.tasks.OnSuccessListener;

import android.Manifest;
import android.location.Location;
import android.net.Uri;
import android.os.Build;
import android.os.Bundle;
import android.content.Intent;
import android.content.pm.PackageManager;
import androidx.core.app.ActivityCompat;
import androidx.fragment.app.FragmentActivity;
import androidx.core.content.ContextCompat;

import android.util.Log;
import android.view.Menu;
import android.view.View;

public class MainActivity extends FragmentActivity {
    private FusedLocationProviderClient mFusedLocationClient;

    private final int MY_PERMISSIONS_REQUEST_ACCESS_LOCATION = 257;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
```

(continues on next page)

(continued from previous page)

```
        setContentView(R.layout.activity_main);

        mFusedLocationClient =
                LocationServices.getFusedLocationProviderClient(this);
        if (Build.VERSION.SDK_INT >= 23) {
            // API level 23 and greater requires permission to get location info.
            Boolean permissionCheck = (ContextCompat.checkSelfPermission(this,
                                       Manifest.permission.ACCESS_FINE_LOCATION)
                                       == PackageManager.PERMISSION_GRANTED);
            if (!permissionCheck) {
                // We have no permission, request to user
                ActivityCompat.requestPermissions(this,
                        new String[]{Manifest.permission.ACCESS_FINE_LOCATION},
                        MY_PERMISSIONS_REQUEST_ACCESS_LOCATION);
            }
        }
    }

    @Override
    public void onRequestPermissionsResult(int requestCode,
                                           String permissions[],
                                           int[] grantResults) {
        switch (requestCode) {
            case MY_PERMISSIONS_REQUEST_ACCESS_LOCATION: {
                if (grantResults.length > 0 && grantResults[0] ==
                    PackageManager.PERMISSION_GRANTED) {
                    // permission is granted
                    int resultCode =
                        GoogleApiAvailability.getInstance()
                                             .isGooglePlayServicesAvailable(this);
                    if (resultCode != ConnectionResult.SUCCESS) {
                        // We cannot use Googleplay service, sample app will
                        // terminate.
                        finish();
                    }
                } else {
                    // permission is not granted, we sample app will terminate.
                    finish();
                }
            }
        }
    }

    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        getMenuInflater().inflate(R.menu.main, menu);
        return true;
    }

    public void onStartMap(View view) {
        Log.d("onStartMap()"," called");
        // *** POINT 1 *** You do not need to display an application privacy policy
        // if your application will only use the information it obtains within the
        // device.
        mFusedLocationClient.getLastLocation()
            .addOnSuccessListener(this, new OnSuccessListener<Location>() {
```

(continues on next page)

(continued from previous page)

```
            @Override
            public void onSuccess(Location location) {
                if (location != null) {
                    Intent intent =
                        new Intent(Intent.ACTION_VIEW,
                            Uri.parse("geo:" + location.getLatitude() +
                                      "," + location.getLongitude())
                        );
                    startActivity(intent);
                }
            }
        });
    }
}
```

Sample description on the marketplace is below.

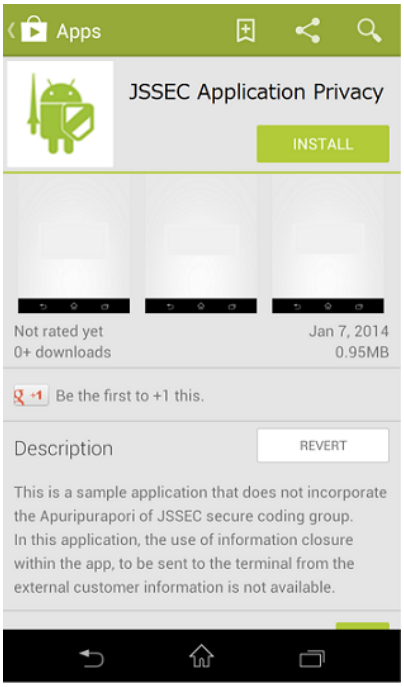


Fig. 5.5.3: Description on the marketplace

5.5.2 Rule Book

When working with private date, obey the following rules.

1. *Restrict transmissions of user data to the minimum necessary (Required)*
2. *On first launch (or application update), obtain broad consent to transmit user data that requires particularly delicate handling or that may be difficult for users to change (Required)*
3. *Obtain specific consent before transmitting user data that requires particularly delicate handling (Required)*
4. *Provide methods by which the user can review the application privacy policy (Required)*
5. *Use UUIDs or cookies for identifiers linked with user data (Do not use device-specific identifiers) (Required)*
6. *Provide methods by which transmitted data can be deleted and transmitting data can be stopped by user operations (Recommended)*

7. *If you will only be using user data within the device, notify the user that data will not be transmitted externally. (Recommended)*
8. *Place a summary version of the application privacy policy in the assets folder (Recommended)*

5.5.2.1 Restrict transmissions of user data to the minimum necessary (Required)

When transmitting usage data to external servers or other destinations, restrict transmissions to the bare minimum necessary to provide service. In particular, you should design that applications have access to only user data of which purpose of use the user can imagine on the basis of the application description.

For example, an application that the user can imagine it is an alarm application, must not have access location data. On the other hand, if an alarm application can sound the alarm depending on the location of user and its feature is written on the description of the application, the application may have access to location data.

In cases where information need only be accessed within an application, avoid transmitting it externally and take other steps to minimize the possibility of inadvertent leakage of user data.

5.5.2.2 On first launch (or application update), obtain broad consent to transmit user data that requires particularly delicate handling or that may be difficult for users to change (Required)

If an application will transmit to external servers any user data that may be difficult for users to change, or any user data that requires particularly delicate handling, the application must obtain advance consent (opt-in) from the user—before the user begins using the application—informing the user of what types of information will be sent, for what purposes, to servers, and whether or not any third-party providers will be involved. More specifically, on first launch the application should display its application privacy policy and confirm that the user has reviewed it and consented. Also, whenever an application is updated in such a way that it now transmits new types of user data to external servers, it must again confirm that the user has reviewed and consented to these changes. If the user does not consent, the application should terminate or otherwise take steps to ensure that all functions requiring the transmission of data are disabled.

These steps serve to guarantee that users understand how their data will be handled when they use an application, providing users with a sense of security and enhancing their trust in the application.

```
MainActivity.java
protected void onStart() {
    super.onStart();

    // (some portions omitted)
    if (privacyPolicyAgreed <= VERSION_TO_SHOW_COMPREHENSIVE_AGREEMENT_ANEW) {
        // *** POINT *** On first launch (or application update),
        // obtain broad consent to transmit user data that will be handled
        // by the application.
        // When the application is updated, it is only necessary to renew
        // the user's grant of broad consent
        // if the updated application will handle new types of user data.
        ConfirmFragment dialog =
                ConfirmFragment.newInstance(R.string.privacyPolicy,
                                            R.string.agreePrivacyPolicy,
                                            DIALOG_TYPE_COMPREHENSIVE_AGREEMENT);
        dialog.setDialogListener(this);
        FragmentManager fragmentManager = getSupportFragmentManager();
        dialog.show(fragmentManager, "dialog");
    }
```

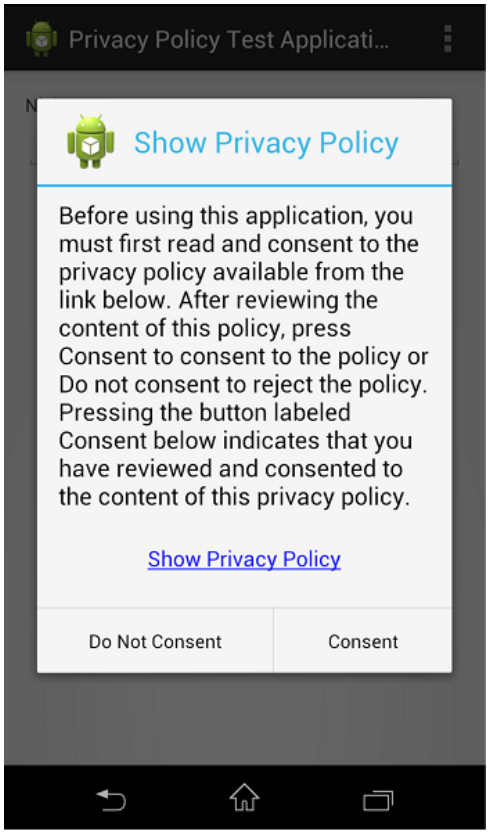


Fig. 5.5.4: Example of broad consent

5.5.2.3 Obtain specific consent before transmitting user data that requires particularly delicate handling (Required)

When transmitting to external servers any user data that requires particularly delicate handling, an application must obtain advance consent (opt-in) from users for each such type of user data (or for each feature that involves the transmission of user data); this is in addition to the need to obtain general consent. If the user does not grant consent, the application must not send the corresponding data to the external server.

This ensures that users can obtain a more thorough understanding of the relationship between an application's features (and the services it provides) and the transmission of user data for which the user granted general consent; at the same time, application providers can expect to obtain user consent on the basis of more precise decision-making.

```
MainActivity.java
public void onSendToServer(View view) {
    // *** POINT *** Obtain specific consent before transmitting user data
    // that requires particularly delicate handling.
    ConfirmFragment dialog =
        ConfirmFragment.newInstance(R.string.sendLocation,
                                    R.string.cofirmSendLocation,
                                    DIALOG_TYPE_PRE_CONFIRMATION);
    dialog.setDialogListener(this);
    FragmentManager fragmentManager = getSupportFragmentManager();
    dialog.show(fragmentManager, "dialog");
}
```

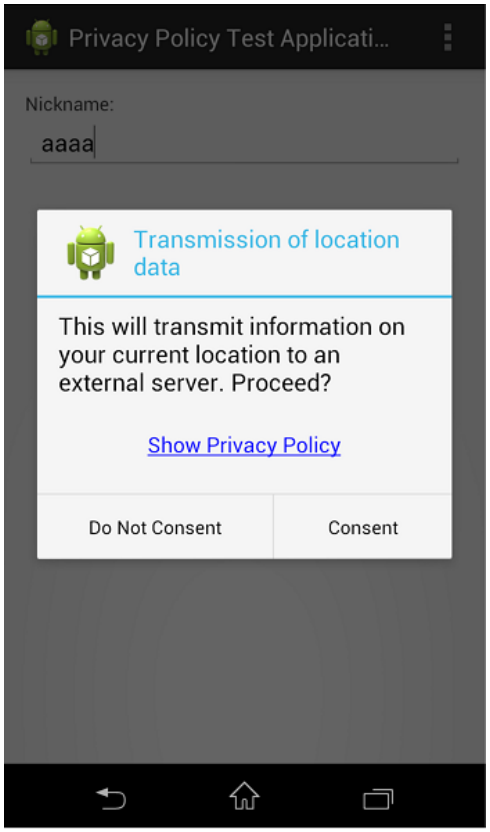


Fig. 5.5.5: Example of specific consent

5.5.2.4 Provide methods by which the user can review the application privacy policy (Required)

In general, the Android application marketplace will provide links to application privacy policies for users to review before choosing to install the corresponding application. In addition to supporting this feature, it is important for applications to provide methods by which users can review application privacy policies after installing applications on their devices. It is particularly important to provide methods by which users can easily review application privacy policies in cases involving consent to transmit user data to external servers to assist users in making appropriate decisions.

```
MainActivity.java
    @Override
    public boolean onOptionsItemSelected(MenuItem item) {
        switch (item.getItemId()) {
        case R.id.action_show_pp:
            // *** POINT *** Provide methods by which the user can review the
            // application privacy policy.
            Intent intent = new Intent();
            intent.setClass(this, WebViewAssetsActivity.class);
            startActivity(intent);
            return true;
```

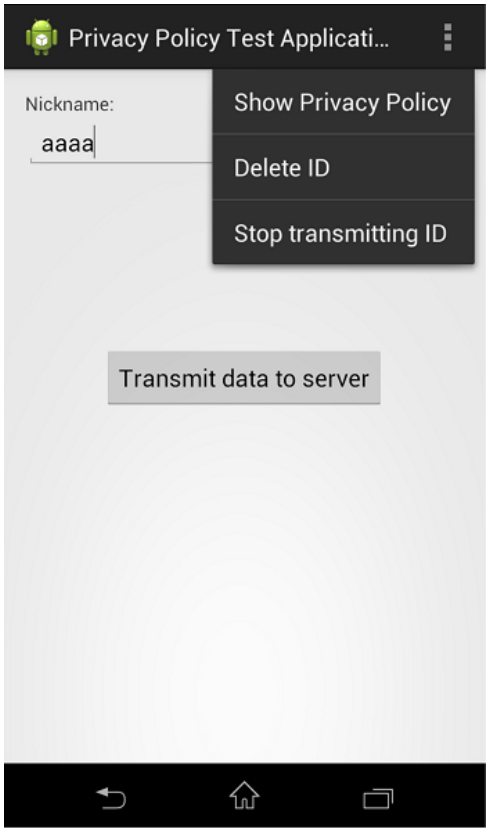


Fig. 5.5.6: Context menu to show privacy policy

5.5.2.5 Use UUIDs or cookies for identifiers linked with user data (Do not use device-specific identifiers) (Required)

IMEIs and other device-specific IDs should not be transmitted in ways that are tied to user data. Indeed, if a device -specific ID and a piece of user data are bundled together and released or leaked to public—even just once—it will be impossible subsequently to change that device -specific ID, whereupon it will be impossible (or at least difficult) to sever ties between the ID and the user data. Also, in Android 10, the obtaining of IMEI and other device-specific identifiers is no longer possible regardless of the targetSdkVersion for the app. For this reason, UUIDs or cookies—that is, variable identifiers that are regenerated each time based on random numbers without using device-specific identifiers—must be transmitted together with user data. This allows an implementation of the notion, discussed below, of the "right to be forgotten."

MainActivity.java

```
    @Override
    protected String doInBackground(String... params) {
        // *** POINT *** Use UUIDs or cookies to keep track of user data
        // In this sample we use an ID generated on the server side
        SharedPreferences sp =
            getSharedPreferences(PRIVACY_POLICY_PREF_NAME, MODE_PRIVATE);
        UserId = sp.getString(ID_KEY, null);
        if (UserId == null) {
            // No token in SharedPreferences; fetch ID from server
            try {
                UserId = NetworkUtil.getCookie(GET_ID_URI, "", "id");
            } catch (IOException e) {
                // Catch exceptions such as certification errors
                extMessage = e.toString();
            }
```

(continues on next page)

(continued from previous page)

```
            // Store the fetched ID in SharedPreferences
            sp.edit().putString(ID_KEY, UserId).commit();

        return UserId;
    }
```

5.5.2.6 Place a summary version of the application privacy policy in the assets folder (Recommended)

It is a good idea to place a summary version of the application privacy policy in the assets folder to ensure that users may review it as necessary. Ensuring that the application privacy policy is present in the assets folder not only allows users to access it easily at any time, but also avoids the risk that users may see a counterfeit or corrupted version of the application privacy policy prepared by a malicious third party.

5.5.2.7 Provide methods by which transmitted data can be deleted and transmitting data can be stopped by user operations (Recommended)

It is a good idea to provide methods by which user data that has been transmitted to external servers can be deleted at the user's request. Similarly, in cases in which the application itself has stored user data (or a copy thereof) within the device, it is a good idea to provide users with methods for deleting this data. And, it is a good idea to provide methods by which transmitting user data can be stopped at the user's request.

This rule (recommendation) is codified by the "right to be forgotten" promoted in the EU; more generally, in the future it seems clear that various proposals will call for further strengthening the rights of users to have their data protected, and for this reason in these guidelines we recommend the provision of methods for the deletion of user data unless there is some specific reason to do otherwise. And, regarding stop transmitting data, it is the one that is defined by the point of view "Do Not Track (deny track)" of the correspondence by the browser is progressing mainly.

```
MainActivity.java
    @Override
    public boolean onOptionsItemSelected(MenuItem item) {
        switch (item.getItemId()) {
        // (some portions omitted)
        case R.id.action_del_id:
            // *** POINT *** Provide methods by which transmitted data can be
            // deleted by user operations.
            new SendDataAsyncTack().execute(DEL_ID_URI, UserId);
            return true;
        }
```

5.5.2.8 If you will only be using user data within the device, notify the user that data will not be transmitted externally. (Recommended)

Even in cases in which user data will only be accessed temporarily within the user's device, it is a good idea to communicate this fact to the user to ensure that the user's understanding of the application's behavior remains full and transparent. More specifically, users should be informed that the user data accessed by an application will only be used within the device for a certain specific purpose and will not be stored or sent. Possible methods for communicating this content to users include specifying it within the description of the application on the application marketplace. Information that is only used temporarily within a device need not be discussed in the application privacy policy.

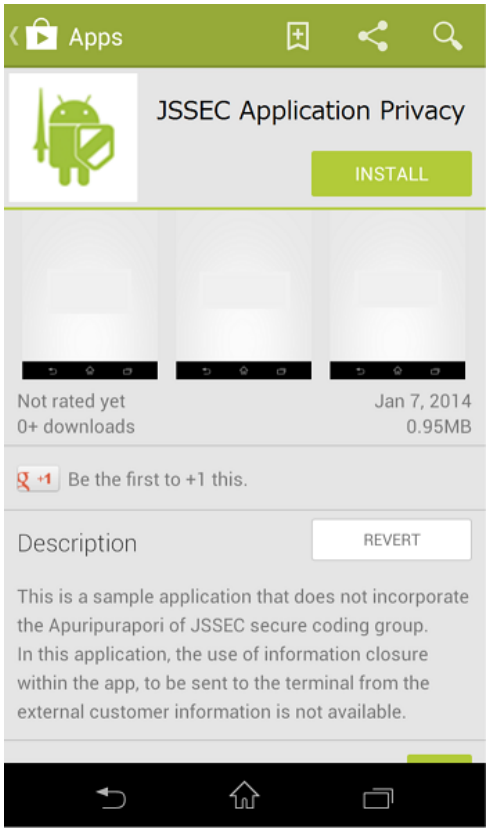


Fig. 5.5.7: Description on the marketplace

5.5.3 Advanced Topics

5.5.3.1 Some background and context regarding privacy policies

For cases in which a smartphone application will obtain user data and transmit this data externally, it is necessary to prepare and display an application privacy policy to inform users of details such as the types of data will be collected and the ways in which the data will be handled. The content that should be included in an application privacy policy is detailed in the Smartphone Privacy Initiative advocated by JMIC's SPI. The primary objective of the application privacy policy should be to state clearly all types of user data that will be accessed by an application, the purposes for which the data will be used, where the data will be stored, and to what destinations the data will be transmitted.

A second document, separate from and required in addition to the application privacy policy, is the Enterprise Privacy Policy, which details how all user data gathered by a corporation from its various applications will be stored, managed, and disposed of. This Enterprise Privacy Policy corresponds to the privacy policy that would traditionally have been prepared to comply with Japan's Personal Information Protection Law.

A detailed description of proper methods for preparing and displaying privacy policies, together with a discussion of the roles played by the various different types of privacy policies, may be found in the document "A Discussion of the Creation and Presentation of Privacy Policies for JSSEC Smartphone Applications", available at https://www.jssec.org/event/20140206/03-1_app_policy.pdf (Japanese only).

5.5.3.2 Glossary of Terms

In the table below we define a number of terms that are used in these guidelines; these definitions are taken from the document "A Discussion of the Creation and Presentation of Privacy Policies for JSSEC Smartphone Applications" (https://www.jssec.org/event/20140206/03-1_app_policy.pdf) (Japanese only).

Table 5.5.1: Glossary of Terms

Term	Description
Enterprise Privacy Policy	A privacy policy that defines a corporation's policies for protecting personal data. Created in accordance with Japan's Personal Information Protection Law.
Application Privacy Policy	An application-specific privacy policy. Created in accordance with the guidelines of the Smartphone Privacy Initiative (SPI) of Japan's Ministry of Internal Affairs and detailed versions containing easily understandable explanations.
Summary version of the Application Privacy Policy	A brief document that concisely summarizes what user information an application will use, for what purpose, and whether or not this information will be provided to third parties.
Detailed version of the Application Privacy Policy	A detailed document that complies with the 8 items specified by the Smartphone Privacy Initiative (SPI) and the Smartphone Privacy Initiative II (SPI II) of Japan's Ministry of Internal Affairs and Communications (MIC).
User data that is easy for users to change	Cookies, UUIDs, etc.
User data that is difficulty for users to change	IMEIs, IMSIs, ICCIDs, MAC addresses, OS-generated IDs, etc.
User data requiring particularly delicate handling	Location information, address books, telephone numbers, email addresses, etc.

5.5.3.3 Version-dependent differences in handling of Android IDs

The Android ID (Settings.Secure.ANDROID_ID) is a randomly-generated 64-bit number expressed as a hexadecimal character string that serves as an identifier to identify individual terminals (although duplicate identifiers are possible in extremely rare cases). For this reason, incorrect usage can create serious risks associated with user tracking, and thus special care must be taken when using Android IDs. However, the rules governing aspects such as ID generation and accessible ranges differ for terminals running Android 7.1 (API Level 25) versus terminals running Android 8.0 (API Level 26). In what follows we describe these differences.

Terminals running Android 7.1(API Level 25) or earlier

For terminals running Android 7.1(API Level 25) or earlier, only one Android ID value exists in a given terminal; this value may be accessed by all apps running on that terminal. However, note that, for terminals with multiuser support, separate values are generated for each user. Android IDs are generated upon the first startup of a terminal after shipping from the factory, and are newly regenerated upon each subsequent factory reset.

Terminals running Android 8.0 (API Level 26) or later

For terminals running Android 8.0 (API Level 26) or later, each app (developer) has its own distinct value, which may only be accessed by the app in question. More specifically, whereas the values used in Android 7.1 (API Level 25) and earlier were user-specific and terminal-specific but not app-specific, in Android 8.0 (API Level 26) and later versions the app signature is added to the list of elements used to generate unique values, so that apps with different signatures now have different Android ID values. (Apps with identical signatures have identical Android ID values.)

The occasions on which Android ID values are generated or modified remain essentially unchanged, but there are a few points to note, as discussed below.

- On package uninstallation / reinstallation:

** **
As long as the signature of the app remains unchanged, its Android ID will be unchanged after uninstalling and reinstalling. On the other hand, note that, if the key used as the signature is modified, the Android ID will be different after re-installation, even if the package name is unchanged.

- On updates to terminals running Android 8.0 (API Level 26) or later:

** **
If an app was already installed on a terminal running Android 7.1 (API Level 25) or earlier, the Android ID

value that may be obtained by the app remains unchanged after the terminal is updated to Android 8.0 (API Level 26) or later. However, this excludes cases in which apps are uninstalled and reinstalled after the update.

Note that all Android IDs are classified as *User information that is difficult for users to exchange* (as described in Section "5.5.3.2. *Glossary of Terms*"), and thus—as noted at the beginning of this discussion—we recommend that similar levels of caution be employed when using Android IDs.

5.5.3.4 Restriction on obtaining non-resettable device identifiers on Android 10

To protect privacy, in Android 10, more restrictions have been placed on the obtaining of non-resettable device identifiers. To obtain device identifiers, the READ_PRIVILEGED_PHONE_STATE permission is required, but this permission is normally not granted to an app. This change affects all apps running in Android 10 regardless of the setting for targetSdkVersion. For this reason, even in apps that ran using the granting of normal permissions, unexpected behavior can still occur due to occurrence of a security exception or returning of null. The types of information that are affected by this and the APIs for obtaining them are as follows. (It is assumed that required permissions such as READ_PHONE_STATE were already granted.)

- Build Class

Table 5.5.2: Build Class

API	Information to be acquired	targetSDKVersion=29	targetSDKVersion29
(field))BUILD.SERIAL	Device serial number	Unknown	Unknown
getSerial	Device serial number	SecurityException	Unknown

- TelephonyManager Class

Table 5.5.3: TelephonyManager Class

API	Information to be acquired	targetSDKVersion=29	targetSDKVersion29
getImei	IMEI	SecurityException	null
getSubscriberId	IMSI	SecurityException	null
getDeviceId	IMEI, MEID, ESN	SecurityException	null
getMeid	MEID	SecurityException	null
getSimSerialNumber	SIM Serial	SecurityException	null

5.5.3.5 Data Access Auditing

"Data access auditing" was added to make the accessing process to user private data such as location information and contact lists transparent on Android 11. To use this, register the AppOpsManager.OnOpNotedCallback instance and implement the callback logic in the component where data access needs to be audited, such as within the onCreate() method of the activity. This enables easy recording of access to user private data.

Defined attribution tags can be included on access records by defining the callback logic to include the defined attribution tag names on the application's log.

```
public class SharePhotoLocationActivity extends AppCompatActivity {
    private Context attributionContext;
    // ~snip~
    @Override
    public void onCreate(@Nullable Bundle savedInstanceState,
                         @Nullable PersistableBundle persistentState) {
        attributionContext = createAttributionContext("sharePhotos");
        // ~snip~
        AppOpsManager.OnOpNotedCallback appOpsCallback =
```

(continues on next page)

(continued from previous page)

```
            new AppOpsManager.OnOpNotedCallback() {
                private void logPrivateDataAccess(String opCode, String␣
↪attributionTag, String trace) {
                    Log.i(TAG, "Private data accessed. ");
                    Log.i(TAG,"Operation:" + opCode);
                    Log.i(TAG,"Attribution Tag: " + attributionTag);
                    Log.i(TAG, "Stack Trace: " + trace);
                }

                @Override
                public void onNoted(@NonNull SyncNotedAppOp syncNotedAppOp) {
                    logPrivateDataAccess(syncNotedAppOp.getOp(),
                                         syncNotedAppOp.getAttributionTag(),
                                         Arrays.toString(new Throwable().
↪getStackTrace()));
                }

                @Override
                public void onSelfNoted(@NonNull SyncNotedAppOp syncNotedAppOp) {
                    logPrivateDataAccess(syncNotedAppOp.getOp(),
                                         syncNotedAppOp.getAttributionTag(),
                                         Arrays.toString(new Throwable().
↪getStackTrace()));
                }

                @Override
                public void onAsyncNoted(@NonNull AsyncNotedAppOp asyncNotedAppOp)
↪{
                    logPrivateDataAccess(asyncNotedAppOp.getOp(),
                                         asyncNotedAppOp.getAttributionTag(),
                                         asyncNotedAppOp.getMessage());
                }
            };

        AppOpsManager appOpsManager = getSystemService(AppOpsManager.class);
        if (appOpsManager != null) {
            appOpsManager.setNotedAppOpsCollector(appOpsCollector);
        }
    }
```

Also, for applications that target Android 12, these attribution tags must be declared within the manifest file.

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    package="xxx.xxxx.myapplication">
    ...
    <attribution
        android:tag="sharePhotos"
        android:label="@string/share_photos_attribution_label" />
    ...
</manifest>
```

If attribution tags that have not been declared are to be used, the attributionTag value is output to LogCat in the null state.

5.5.3.6 Location Access

Location access in Android OS has undergone continuous refinement and tightening of its permission model aimed at privacy protection with each OS version, starting with the separation of foreground and background in Android 10, incremental permission requests (Android 11), selection of approximate location (Android 12), and separation of Wi-Fi device detection permissions (Android 13).

However, under conventional permission models, a three-choice dialog offering "While using the app," "Only this time," and "Don't allow" was displayed every time location information was requested. Forcing users to make a decision every time imposed a cognitive load, becoming a cause of "dialog fatigue."

Introduction of the "Location Button" in Android 17

The "Location Button" newly introduced in Android 17 is a groundbreaking feature that balances user experience (UX) and privacy protection. By shifting to a method where users themselves tap a "Location Button" placed on the screen, expression of intent and permission grant are completed in one step, dramatically improving operational smoothness.

The greatest benefit brought by this feature is the shift from "always allow" to "data minimization." Tapping the button just once grants precise location access for that time only. For operations such as "tagging a location in a social media post" or "searching for nearby stores," permission automatically expires once the task is completed, giving users a strong sense of security that "the app is not tracking in the background without permission."

Because the Location Button is provided as an Android Jetpack library from Google, it can be easily placed in existing screen layouts. It also possesses high backward compatibility; when a user taps this button on a device with Android 16 or lower, a conventional location request dialog is automatically displayed. Developers do not need to implement complex conditional branching for each OS version, making it possible to achieve both modern UI and conventional behavior with simple code.

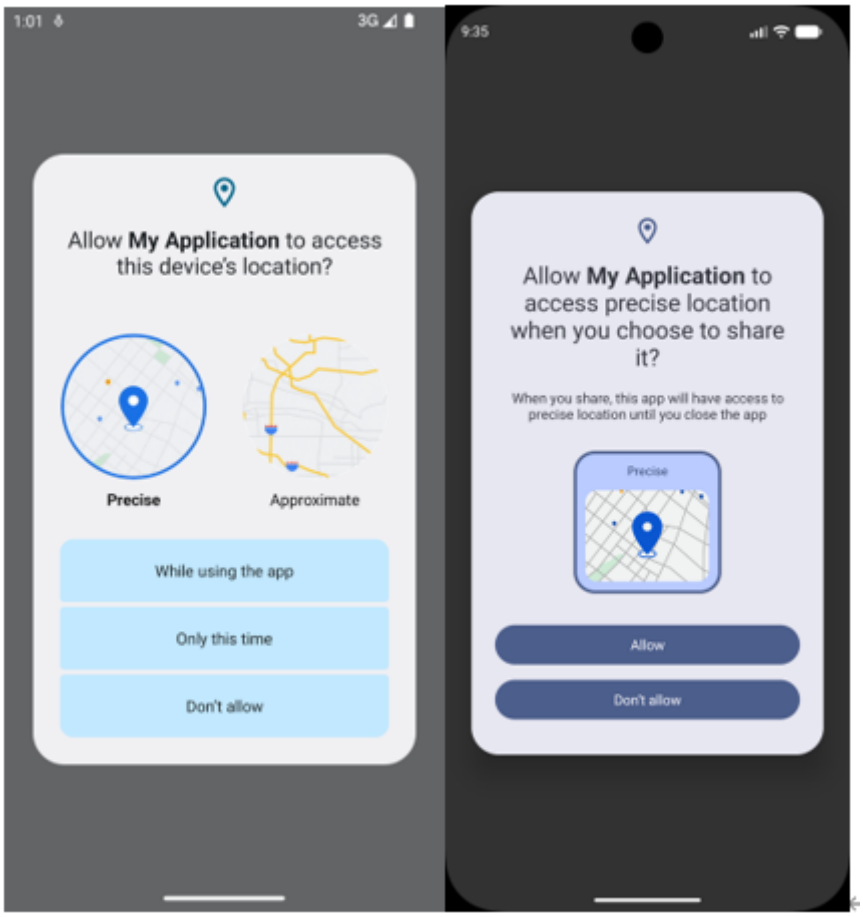


Fig. 5.5.8: Screen when operated on a device with Android 16 or lower (left), Screen when operated on an Android 17 device (right)

5.5.3.7 Local Network Access Permission

Previously, as long as an app held the INTERNET permission, it could freely access devices on a home network, but this carried risks such as inferring user environment and usage status, personal identification, and hijacking IoT devices. Behind this specification change is the purpose of protecting user privacy and security from such unauthorized access. Local network restrictions prevent malicious apps from using unrestricted LAN access to secretly conduct user tracking or fingerprinting, preventing apps from communicating within a LAN unless explicitly permitted by the user.

Local Network Restrictions in Android 16 (Opt-in)

In Android 16, a restriction was introduced where the NEARBY_WIFI_DEVICES permission is required when an app communicates with devices on a local network (LAN). The scope of this permission, originally introduced in Android 13, has been expanded, and access to TCP/UDP communications to IP addresses on a LAN (such as 192.168.x.x or 10.x.x.x) as well as local network service discovery such as mDNS and NSD will not be possible without the permission. However, this restriction is turned off by default and is applied only when a developer explicitly enables the RESTRICT_LOCAL_NETWORK flag using an adb command (opt-in method).

Mandatory Local Network Permission in Android 17

In Android 17 and later, this restriction is forcibly applied to all apps. A new ACCESS_LOCAL_NETWORK runtime permission has been introduced, making requests mandatory. When an app communicates with devices on the same Wi-Fi (such as smart home devices, media casting, file sharing, etc.), it needs to declare this permission and obtain user permission at runtime. Note that because this permission belongs to the existing NEARBY_DEVICES permission group, users who have already granted other NEARBY_DEVICES permissions will not be shown a prompt requesting permission grant again.

Since it is expected to be forcibly applied to all apps in the future, apps that require LAN access need to handle and verify it early.

Implementation and Verification Method

First, declare permissions in AndroidManifest.xml such as NEARBY_WIFI_DEVICES for Android 16, ACCESS_LOCAL_NETWORK for Android 17, and INTERNET.

```
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />

<!-- For Android 16 (For opt-in verification) -->
<uses-permission android:name="android.permission.NEARBY_WIFI_DEVICES"
    android:usesPermissionFlags="neverForLocation"
    tools:targetApi="s" />

<!-- Android 17 or higher runtime permission -->
<uses-permission android:name="android.permission.ACCESS_LOCAL_NETWORK" />
```

When verifying local network restrictions in an Android 16 environment, enable the restriction via adb command and reboot the device.

```
adb shell am compat enable RESTRICT_LOCAL_NETWORK <package_name>
adb reboot
```

On the app side, connection can be verified with code like the following using Jetpack Compose, for example. Pressing the button attempts a socket connection to an IP address on the LAN and displays the result on the screen.

```
Button(
    onClick = {
        testResult = "Testing..."
        Thread {
            try {
                Socket().use { socket ->
                    socket.connect(InetSocketAddress("192.168.3.1", 80), 2000)
                }
                (context as? ComponentActivity)?.runOnUiThread {
                    testResult = "Connection successful"
                }
            } catch (e: Exception) {
                (context as? ComponentActivity)?.runOnUiThread {
                    testResult = "Connection failed: ${e.localizedMessage}"
                }
            }
        }.start()
    },
    modifier = Modifier.fillMaxWidth()
) {
    Text("Local Network Connection Test")
}
```

If you press this button while permission is denied (or when opted in on Android 16 without granting permission), errors such as sendto failed: EPERM (Operation not permitted) or sendto failed: ECONNABORTED (Operation not permitted) are returned. Appropriately requesting and obtaining permission at runtime restores LAN access, allowing normal communication.

5.5.3.8 Auto-hibernation Function for Unused Applications on Android 12

If Android 12 (API Level 31) is the target, the auto-hibernation function is applied to applications that have not been used for a certain period, in addition to the permission auto-reset function introduced in Android 11. Auto-hibernation function has the following characteristics.

- All files within the application cache are deleted and optimization is performed not based on the performance, but based on the storage capacity
- The application will not be able to run jobs and alerts in the background
- The application will not be able to receive push notifications (e.g. high priority messages sent through Firebase Cloud Messaging)

The hibernation state will end when the user performs operations on the application. However, jobs, alerts, and notifications that had been scheduled before the application enters the hibernation state must have their schedules set again.

For applications where inconveniences may occur due to auto-reset of permissions and the auto-hibernation function, such as applications that periodically synchronize data between devices and servers, users can exclude them by turning the “Remove permissions and free up space” option off.

To experimentally switch the application to the hibernation state, perform the following commands. Doing so will simulate the hibernation state.

- Enable the hibernation state behavior

```
$ adb shell device_config put app_hibernation app_hibernation_enabled true
```

- Forcibly set the application to the hibernation state

```
$ adb shell cmd app_hibernation set-state org.jssec.android.activity.
↪privateactivity true
```

5.5.3.9 API Return Value Change Following Specification Changes to the Package Access

If Android 12 is installed to the device and if Android 11.0 (API Level 30) or later is specified, the return values of the following methods become filtered values based on the specification changes of the package access. This complies with the minimum permission principle introduced on the package access of Android 11.

- getAllPermissionGroups()
- getPermissionGroupInfo()
- getPermissionInfo()
- queryPermissionsByGroup()

To verify the operation capability, a custom permission group is created as shown below, and a comparison was made on how the getAllPermissionGroups() values change.

```
<permission-group android:name="android.permission-group.JSSEC"
    android:label="@string/perm_label"
    android:icon="@drawable/ic_launcher_foreground"
    android:description="@string/perm_description"
    android:permissionGroupFlags="personalInfo"
    android:priority="360"/>
```

- If Android 11 is installed to the device and if Android 11.0 (API Level 30) is specified

```
com.google.android.gms.permission.CAR_INFORMATION
android.permission-group.CONTACTS
android.permission-group.PHONE
```

(continues on next page)

(continued from previous page)

```
android.permission-group.CALENDAR
android.permission-group.CALL_LOG
android.permission-group.CAMERA
android.permission-group.UNDEFINED
android.permission-group.ACTIVITY_RECOGNITION
android.permission-group.SENSORS
android.permission-group.LOCATION
android.permission-group.STORAGE
android.permission-group.MICROPHONE
android.permission-group.SMS
android.permission-group.JSSEC
```

- If Android 12 is installed to the device and if Android 11.0 (API Level 30) is specified

```
com.google.android.gms.permission.CAR_INFORMATION
android.permission-group.CONTACTS
android.permission-group.PHONE
android.permission-group.CALENDAR
android.permission-group.CALL_LOG
android.permission-group.CAMERA
android.permission-group.UNDEFINED
android.permission-group.ACTIVITY_RECOGNITION
android.permission-group.SENSORS
android.permission-group.LOCATION
android.permission-group.STORAGE
android.permission-group.MICROPHONE
android.permission-group.SMS
```

You can see that the custom permission group could not be acquired if Android 12 is installed to the device and if Android 11.0 (API Level 30) is specified. The list of the packages installed to the device is based on the concept of privacy. If the application needs to access other applications, it is necessary to clearly specify other applications in <queries>. <queries> is also used on various sample codes of this guide.

5.5.3.10 Microphones and Cameras For Android 12

Microphones and cameras can be set to on or off from quick settings or the [Privacy] screen of system settings on Android 12. These settings are reflected on all applications on the device.

If users launch the application that accesses microphones and cameras that have been set to off, the system notifies users that these devices are off.

This function is available for use only on supported devices and can be confirmed with the following code.

```
SensorPrivacyManager sensorPrivacyManager = getApplicationContext().
↪getSystemService(SensorPrivacyManager.class);
boolean supportsMicrophoneToggle = sensorPrivacyManager.
↪supportsSensorToggle(Sensors.MICROPHONE);
boolean supportsCameraToggle = sensorPrivacyManager.supportsSensorToggle(Sensors.
↪CAMERA);
```

For applications that use microphones and cameras, it is recommended to access them through the following methods based on permissions.

- The device camera will not be accessed until the user applies CAMERA permissions to the application.
- The device microphone will not be accessed until the user applies RECORD_AUDIO permissions to the application.

When the application accesses the microphone or camera, an icon is displayed in the status bar to indicate that it is being accessed. Users can confirm which application is currently using microphones or cameras by tapping the icon from quick settings.

5.5.3.11 SameSite Cookie on WebView

The following privacy protection changes are applied on WebView for applications that target Android 12.

- Cookies without a SameSite attribute are handled as SameSite=Lax. In other words, only cookies of the same domain as the accessed website can be set.
- The Secure attribute must be clearly specified for cookies with SameSite=None. In other words, unless an HTTPS connection is established, cookie settings and loading is unavailable.

This is to prevent Cross Site Request Forgery (CSRF) attacks and is applied on application WebView targeted for Android 12 and later.

If using WebView on an application or if managing websites and services that use cookies, it is necessary to check that the existing flow operates correctly in advance.

Furthermore, this change is applied on WebView version (89.0.4385.0) and later of Android 12, and version verification can be performed from the "App Info" screen ("Settings" - "Applications" - "Android System WebView")[25].

5.5.3.12 Partial Screen Sharing

Android 15 introduces a new partial screen sharing feature, which allows you to share or record only specific app windows. This change aims to address the need for privacy protection and focusing on the content of a specific application. Below are the steps on how to implement it.

1. Request permission for overlay display

The app needs to obtain permission to display an overlay on top of other apps. The following code requests the user's permission for overlay display.

```
private fun checkOverlayPermissionAndRequestMediaProjection() {
    if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.M) {
        if (!Settings.canDrawOverlays(this)) {
            val intent = Intent(Settings.ACTION_MANAGE_OVERLAY_PERMISSION, Uri.
↪parse("package:$packageName"))
            overlayPermissionLauncher.launch(intent)
        } else {
            requestMediaProjectionPermission()
        }
    } else {
        requestMediaProjectionPermission()
    }
}

private val overlayPermissionLauncher = registerForActivityResult(
    ActivityResultContracts.StartActivityForResult()
) { result ->
    if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.M) {
        if (Settings.canDrawOverlays(this)) {
            requestMediaProjectionPermission()
        } else {
            Toast.makeText(this, "Overlay permission denied", Toast.LENGTH_SHORT).
↪show()
        }
```

(continues on next page)

[25] The WebView version was 93.0.4577.82 when verified on an Android 12 (build number SPB5.210812.002) terminal

(continued from previous page)

```
    }
}
```

When the above code is executed, a dialog appears requesting the user's permission for the application to display an overlay on top of other applications.

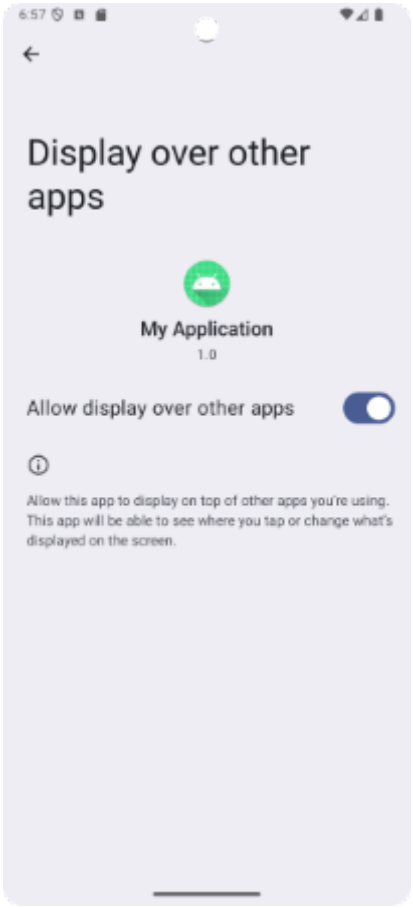


Fig. 5.5.9: Permission to display overlays

2. Request permission to capture the screen and handle the result

Once overlay permission has been granted, use MediaProjectionManager to request permission to capture the screen and handle the result.

```
private fun requestMediaProjectionPermission() {
    val captureIntent = mediaProjectionManager.createScreenCaptureIntent()
    mediaProjectionLauncher.launch(captureIntent)
}

private val mediaProjectionLauncher = registerForActivityResult(
    ActivityResultContracts.StartActivityForResult()
) { result ->
    if (result.resultCode == Activity.RESULT_OK && result.data != null) {
        val serviceIntent = Intent(this, MediaProjectionService::class.java).apply
↪{
            putExtra("resultCode", result.resultCode)
            putExtra("data", result.data)
        }
```

(continues on next page)

(continued from previous page)

```
        if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.O) {
            startForegroundService(serviceIntent)
        } else {
            startService(serviceIntent)
        }
    } else {
        Toast.makeText(this, "Permission denied", Toast.LENGTH_SHORT).show()
    }
}
```

When the above code is executed, a dialog will appear asking the user for permission for the application to capture the screen of other apps.

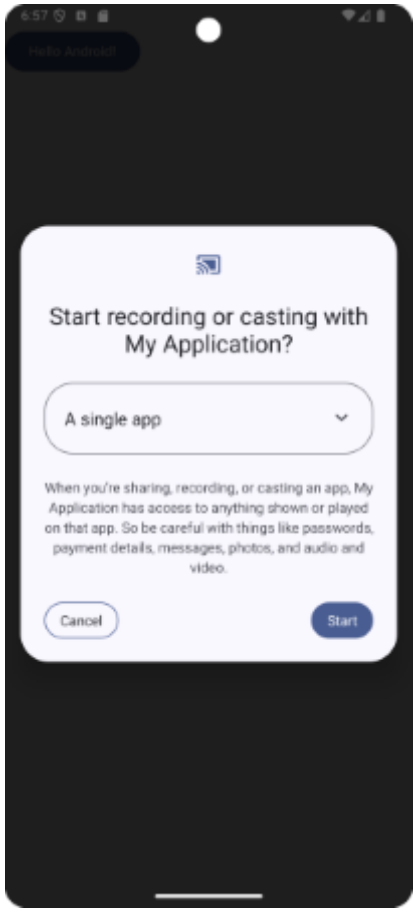


Fig. 5.5.10: Allow recording or casting

3. Select the app you want to capture

When you click the Start button, a list of apps will be displayed and you can select the app you want to capture. The following is an example of the Android standard Clock app.

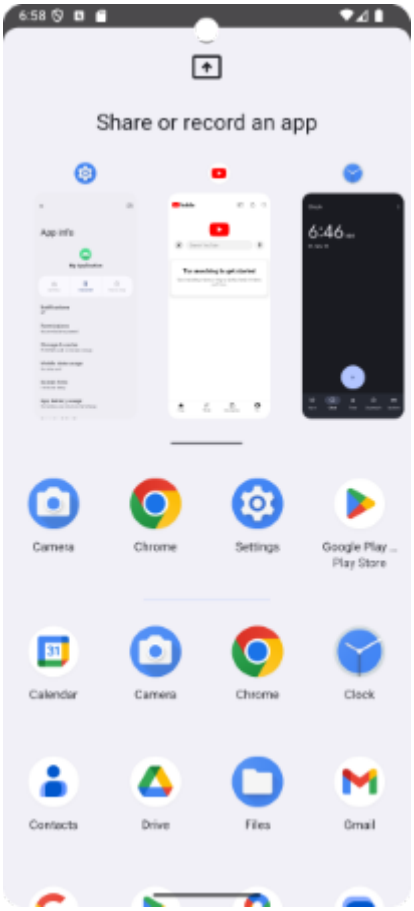


Fig. 5.5.11: Apps list

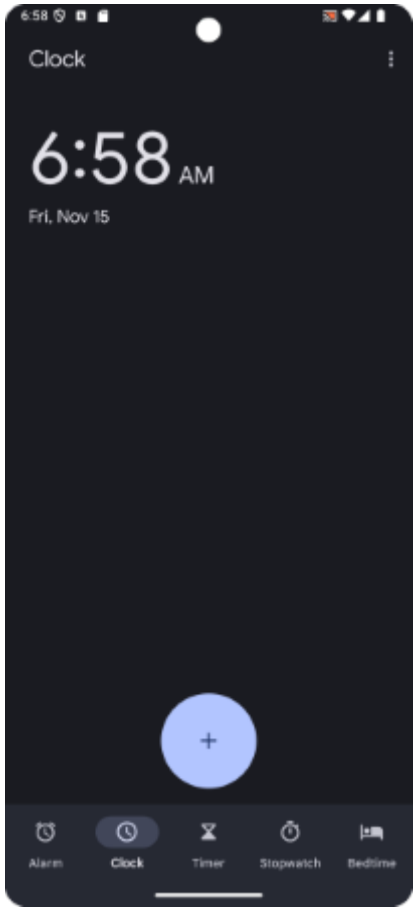


Fig. 5.5.12: Select the Clock app

5.5.3.13 Changes in Global State Management in DND mode

Before Android 14, apps could directly modify DND global states using setInterruptionFilter and setNotificationPolicy.

```
fun setInterruptionFilter(context: Context, filter: Int) {
    val notificationManager = context.getSystemService(Context.NOTIFICATION_
↪SERVICE) as NotificationManager
    notificationManager.setInterruptionFilter(filter)
}
```

Starting in Android 15, apps can no longer directly modify DND global states. Instead, they must use AutomaticZenRule, which automatically enables/disables DND mode when specific conditions (time schedules, location triggers, calendar events) are met. The system merges existing rules by prioritizing the most restrictive policy, preventing unauthorized apps from overriding user settings.

AutomaticZenRule is a system-managed rule that enables indirect control of DND mode by activating/deactivating it based on predefined conditions.

Creating Rules

```
fun createAutomaticZenRule(context: Context) {
    val notificationManager = context.getSystemService(Context.NOTIFICATION_
↪SERVICE) as NotificationManager

    // Define the condition for the rule (e.g., a condition to enable DND at a␣
↪specific time)
```

(continues on next page)

(continued from previous page)

```
    val conditionId = Uri.parse("content://com.example.myapp/condition")

    // Create the AutomaticZenRule
    val zenRule = AutomaticZenRule(
        "My Automatic Zen Rule",
        ComponentName(context.packageName, MyConditionConfigurationActivity::class.
↪java.name),
        conditionId,
        null, // ZenPolicy is optional, pass null if not needed
        NotificationManager.INTERRUPTION_FILTER_PRIORITY,
        true
    )

    // Add the rule
    val ruleUri = notificationManager.addAutomaticZenRule(zenRule)
    if (ruleUri != null) {
        println("AutomaticZenRule created: $ruleUri")
    } else {
        println("Failed to create AutomaticZenRule")
    }
}
```

Updating Rules

```
fun updateAutomaticZenRule(context: Context, ruleId: String) {
    val notificationManager = context.getSystemService(Context.NOTIFICATION_
↪SERVICE) as NotificationManager

    // Change the condition for the rule
    val newConditionId = Uri.parse("content://com.example.myapp/new_condition")

    // Retrieve the existing rule
    val zenRule = notificationManager.getAutomaticZenRule(ruleId)
    if (zenRule != null) {
        // Update the rule
        val updatedZenRule = AutomaticZenRule(
            zenRule.name,
            zenRule.owner,
            zenRule.configurationActivity,
            newConditionId,
            zenRule.zenPolicy,
            zenRule.interruptionFilter,
            zenRule.isEnabled
        )

        notificationManager.updateAutomaticZenRule(ruleId, updatedZenRule)
        println("AutomaticZenRule updated: $ruleId")
    } else {
        println("AutomaticZenRule not found: $ruleId")
    }
}
```

Calling legacy APIs that previously affected global state (setInterruptionFilter, setNotificationPolicy) now implicitly creates or updates an AutomaticZenRule. This rule toggles on/off based on the API call cycle.

For example, using setInterruptionFilter(NotificationManager.INTERRUPTION_FILTER_ALL) to disable DND mode will also implicitly create an AutomaticZenRule, which the system manages appropriately.

5.5.3.14 Private Space

Android 15 implements the Private Space feature. Private Space creates a secure, isolated environment on the device, allowing apps and data to be managed privately. This space is created in a way that is tied to the device's primary user, and the user decides whether to use it or install apps. Therefore, app developers generally do not need to implement any special APIs or make modifications, but it is recommended that they be aware of the following specifications.

- Private Spaces are based on Android's multi-user model and operate as separate user profiles. Apps are installed within these profiles, and data is also isolated. However, additional protections such as encryption are not automatically applied, so the same measures as for regular applications are required.
- Regarding storage, an external storage area is provided for private spaces and is allocated a separate area from the main user. As a rule, apps in each area cannot directly access files stored in the external storage of other areas. When unlocked, files can only be exchanged via Sharesheet or the system gallery picker.
- App behavior varies greatly depending on the lock status of the private space. When locked, all apps in the private space are completely stopped, and notifications and background processing are not displayed. In addition, notifications and the existence of the apps themselves are hidden from the launcher, settings screen, etc. Notifications and apps are only displayed and available after unlocking.
- In this way, private spaces not only provide storage isolation but also protect user privacy in many ways, including limiting notifications, app visibility, and background operation. In particular, apps that require background operation or constant notifications (e.g., medical apps) may not be recommended for use within private spaces.
- You can use different Google accounts in the main space and private space.

As such, private spaces have a wide range of specifications, not just for storage, but also for notifications, app behavior, visibility, etc. Measures that focus only on storage are insufficient, so it is advisable for app developers to understand the overall picture of private spaces and consider how to respond and provide guidance to users as needed.

5.5.3.15 Transition to Contact Picker and Proper Handling of Contact Data

With the tightening of Contacts Provider (CP2) in Android 17, a model where "contacts are shared only as much as selected by the user" is recommended. Full access to contacts, which was previously performed using the READ_CONTACTS permission, is significantly restricted, making user-selected access via Contact Picker the default. Instead of having full access to contacts, apps need to transition to a design that uses Contact Picker to retrieve only the data individually selected and permitted by the user (phone numbers, email addresses, names, addresses, company names/titles, etc.).

Behind this restriction is a review of over-collection of information by apps. Policy has shifted from traditional persistent full access to temporary access restricted to the necessary minimum. If full access to contacts is absolutely necessary, as in the case of calling apps or contact management apps, it will be subject to review, requiring declaration of a legitimate reason in Google Play Console.

Furthermore, information obtained through Contact Picker is intended for temporary use, and unnecessary long-term storage or continuous background usage should be avoided. Saving or uploading contact data without explicitly informing the user may be considered a violation of User Data Policy. The purpose of use for retrieved data must be limited, and even when saving data cannot be avoided, its usage and scope must be clearly explained to the user in advance.

From the perspective of user experience (UX) as well, implementations that suddenly request permissions or data provision upon app launch should be avoided. Explaining clearly why data is needed before retrieving it and designing Contact Picker to be launched triggered by explicit user operation is important in protecting privacy and gaining trust from users.

For implementation, in Android 17 or higher environments, calls using existing Intent.ACTION_PICK are automatically upgraded to the new Contact Picker interface. However, to utilize the full capabilities of the new Contact Picker, such as specifying multiple data types simultaneously, explicitly using the newly established Intent.ACTION_PICK_CONTACTS is recommended.

```
// Example of launching Contact Picker recommended in Android 17 and later
val intent = Intent(Intent.ACTION_PICK_CONTACTS)

// *Specify data types to retrieve (such as phone numbers or email addresses) in␣
↪extras as needed
startActivityForResult(intent, REQUEST_CODE)
```

5.6 Using Cryptography

In the security world, the terms "confidentiality", "integrity", and "availability" are used in analyzing responses to threats. These three terms refer, respectively, to measures to prevent the third parties from viewing private data, protections to ensure that the data referenced by users has not been modified (or techniques for detecting when it has been falsified) and the ability of users to access services and data at all times. All three of these elements are important to consider when designing security protections. In particular, encryption techniques are frequently used to ensure confidentiality and integrity, and Android is equipped with a variety of cryptographic features to allow applications to realize confidentiality and integrity.

In this section we will use sample code to illustrate methods by which Android applications can securely implement encryption and decryption (to ensure confidentiality) and message authentication codes (MAC) or digital signatures (to ensure integrity).

5.6.1 Sample Code

A variety of cryptographic methods have been developed for specific purposes and conditions, including use cases such as "encrypting and decrypting data (to ensure confidentiality)" and "detecting falsification of data (to ensure integrity)". Here is sample code that is categorized into three broad groups of cryptography techniques on the basis of the purpose of each technology. The features of the cryptographic technology in each case should make it possible to choose an appropriate encryption method and key type. For cases in which more detailed considerations are necessary, see Section "5.6.3.1. *Choosing encryption methods*".

- Protecting data from third-party eavesdropping

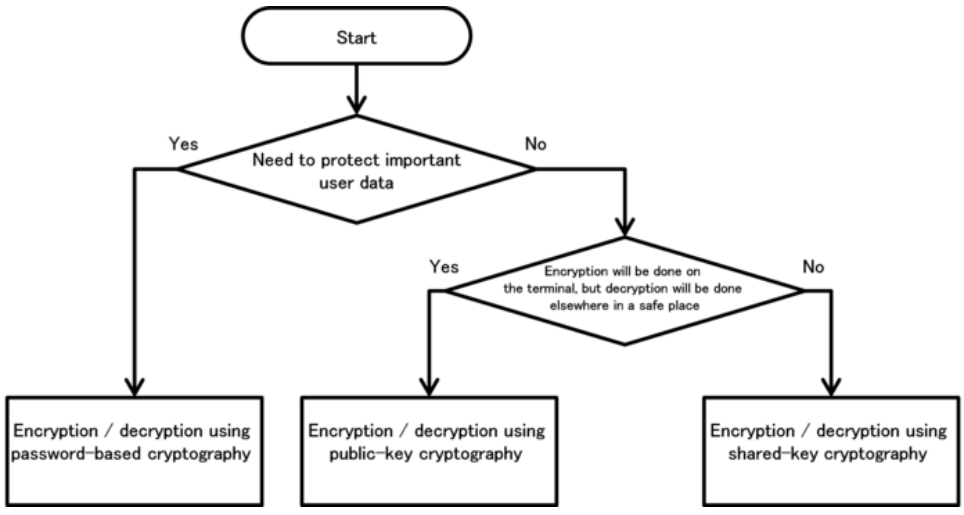


Fig. 5.6.1: Selection flowchart for sample code to protect data from eavesdropping

- Detecting falsification of data made by a third party

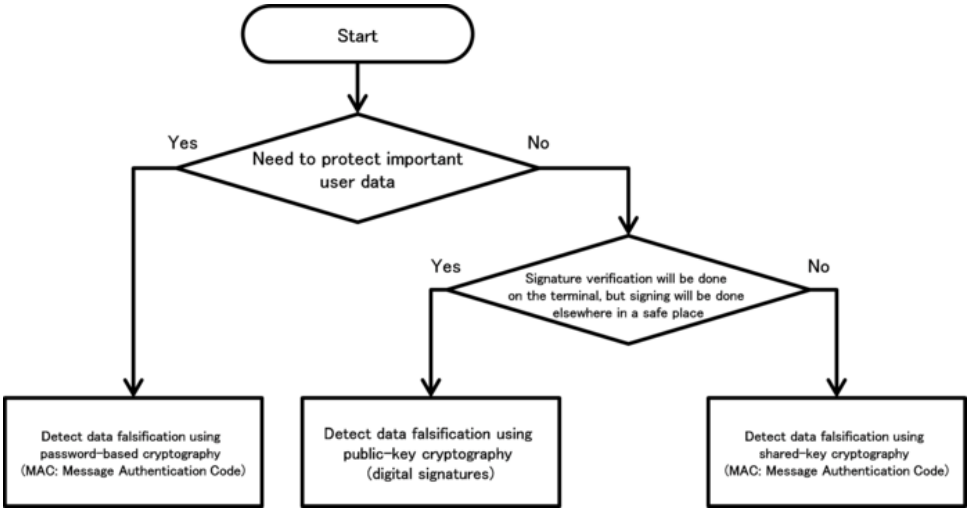


Fig. 5.6.2: Selection flowchart for sample code to detect falsifications

5.6.1.1 Encrypting and Decrypting With Password-based Keys

You may use password-based key encryption for the purpose of protecting a user's confidential data assets.

Points:

1. Explicitly specify the encryption mode and the padding.
2. Use strong encryption technologies (specifically, technologies that meet the relevant criteria), including algorithms, block cipher modes, and padding modes.
3. When generating a key from password, use Salt.
4. When generating a key from password, specify an appropriate hash iteration count.
5. Use a key of length sufficient to guarantee the strength of encryption.

AesCryptoPBEKey.java

```
/*
 * Copyright (C) 2012-2026 Japan Smartphone Security Association
 *
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 *
 *      http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */
package org.jssec.android.cryptsymmetricpasswordbasedkey;

import java.security.InvalidAlgorithmParameterException;
import java.security.InvalidKeyException;
import java.security.NoSuchAlgorithmException;
import java.security.SecureRandom;
import java.security.spec.InvalidKeySpecException;
import java.util.Arrays;
```

(continues on next page)

(continued from previous page)

```
import javax.crypto.BadPaddingException;
import javax.crypto.Cipher;
import javax.crypto.IllegalBlockSizeException;
import javax.crypto.NoSuchPaddingException;
import javax.crypto.SecretKey;
import javax.crypto.SecretKeyFactory;
import javax.crypto.spec.IvParameterSpec;
import javax.crypto.spec.PBEKeySpec;

public final class AesCryptoPBEKey {

    // *** POINT 1 *** Explicitly specify the encryption mode and the padding.
    // *** POINT 2 *** Use strong encryption technologies (specifically,
    // technologies that meet the relevant criteria),
    // including algorithms, block cipher modes, and padding modes.
    // Parameters passed to the getInstance method of the Cipher class:
    // Encryption algorithm, block encryption mode, padding rule
    // In this sample, we choose the following parameter values:
    // encryption algorithm=AES, block encryption mode=CBC, padding
    // rule=PKCS7Padding
    private static final String TRANSFORMATION = "AES/CBC/PKCS7Padding";

    //  A string used to fetch an instance of the class that generates the key
    private static final String KEY_GENERATOR_MODE =
            "PBEWITHSHA256AND128BITAES-CBC-BC";

    // *** POINT 3 *** When generating a key from a password, use Salt.
    // Salt length in bytes
    public static final int SALT_LENGTH_BYTES = 20;

    // *** POINT 4 *** When generating a key from a password,
    // specify an appropriate hash iteration count.
    // Set the number of mixing repetitions used when generating keys via PBE
    private static final int KEY_GEN_ITERATION_COUNT = 1024;

    // *** POINT 5 *** Use a key of length sufficient to guarantee
    // the strength of encryption.
    // Key length in bits
    private static final int KEY_LENGTH_BITS = 128;

    private byte[] mIV = null;
    private byte[] mSalt = null;

    public byte[] getIV() {
        return mIV;
    }

    public byte[] getSalt() {
        return mSalt;
    }

    AesCryptoPBEKey(final byte[] iv, final byte[] salt) {
        mIV = iv;
        mSalt = salt;
    }
```

(continues on next page)

(continued from previous page)

```
AesCryptoPBEKey() {
    mIV = null;
    initSalt();
}

private void initSalt() {
    mSalt = new byte[SALT_LENGTH_BYTES];
    SecureRandom sr = new SecureRandom();
    sr.nextBytes(mSalt);
}

public final byte[] encrypt(final byte[] plain, final char[] password) {
    byte[] encrypted = null;

    try {
        // *** POINT 1 *** Explicitly specify the encryption mode and the
        // padding.
        // *** POINT 2 *** Use strong encryption technologies (specifically,
        // technologies that meet the relevant criteria),
        // including algorithms, modes, and padding.
        Cipher cipher = Cipher.getInstance(TRANSFORMATION);

        // *** POINT 3 *** When generating keys from passwords, use Salt.
        SecretKey secretKey = generateKey(password, mSalt);
        cipher.init(Cipher.ENCRYPT_MODE, secretKey);
        mIV = cipher.getIV();

        encrypted = cipher.doFinal(plain);
    } catch (NoSuchAlgorithmException e) {
    } catch (NoSuchPaddingException e) {
    } catch (InvalidKeyException e) {
    } catch (IllegalBlockSizeException e) {
    } catch (BadPaddingException e) {
    } finally {
    }

    return encrypted;
}

public final byte[] decrypt(final byte[] encrypted, final char[] password) {
    byte[] plain = null;

    try {
        // *** POINT 1 *** Explicitly specify the encryption mode and the
        // padding.
        // *** POINT 2 *** Use strong encryption technologies (specifically,
        // technologies that meet the relevant criteria),
        // including algorithms, block cipher modes, and padding modes.
        Cipher cipher = Cipher.getInstance(TRANSFORMATION);

        // *** POINT 3 *** When generating a key from a password, use Salt.
        SecretKey secretKey = generateKey(password, mSalt);
        IvParameterSpec ivParameterSpec = new IvParameterSpec(mIV);
        cipher.init(Cipher.DECRYPT_MODE, secretKey, ivParameterSpec);
```

(continues on next page)

(continued from previous page)

```
            plain = cipher.doFinal(encrypted);
        } catch (NoSuchAlgorithmException e) {
        } catch (NoSuchPaddingException e) {
        } catch (InvalidKeyException e) {
        } catch (InvalidAlgorithmParameterException e) {
        } catch (IllegalBlockSizeException e) {
        } catch (BadPaddingException e) {
        } finally {
        }

        return plain;
    }

    private static final SecretKey generateKey(final char[] password,
                                               final byte[] salt) {
        SecretKey secretKey = null;
        PBEKeySpec keySpec = null;

        try {
            // *** POINT 2 *** Use strong encryption technologies (specifically,
            // technologies that meet the relevant criteria),
            // including algorithms, block cipher modes, and padding modes.
            // Fetch an instance of the class that generates the key
            // In this example, we use a KeyFactory that uses SHA256
            // to generate AES-CBC 128-bit keys.
            SecretKeyFactory secretKeyFactory =
                    SecretKeyFactory.getInstance(KEY_GENERATOR_MODE);

            // *** POINT 3 *** When generating a key from a password, use Salt.
            // *** POINT 4 *** When generating a key from a password,
            // specify an appropriate hash iteration count.
            // *** POINT 5 *** Use a key of length sufficient to guarantee
            // the strength of encryption.
            keySpec = new PBEKeySpec(password,
                    salt,
                    KEY_GEN_ITERATION_COUNT,
                    KEY_LENGTH_BITS);
            // Clear password
            Arrays.fill(password, '?');
            // Generate the key
            secretKey = secretKeyFactory.generateSecret(keySpec);
        } catch (NoSuchAlgorithmException e) {
        } catch (InvalidKeySpecException e) {
        } finally {
            keySpec.clearPassword();
        }

        return secretKey;
    }
}
```

5.6.1.2 Encrypting and Decrypting With Public Keys

In some cases, only data encryption will be performed -using a stored public key- on the application side, while decryption is performed in a separate safe location (such as a server) under a private key. In cases such as this, it is possible to use public-key encryption.

Points:

1. Explicitly specify the encryption mode and the padding
2. Use strong encryption methods (specifically, technologies that meet the relevant criteria), including algorithms, block cipher modes, and padding modes.
3. Use a key of length sufficient to guarantee the strength of encryption.

RsaCryptoAsymmetricKey.java

```
/*
 * Copyright (C) 2012-2026 Japan Smartphone Security Association
 *
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 *
 *      http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */

package org.jssec.android.cryptasymmetrickey;

import java.security.InvalidKeyException;
import java.security.KeyFactory;
import java.security.NoSuchAlgorithmException;
import java.security.PrivateKey;
import java.security.PublicKey;
import java.security.interfaces.RSAPublicKey;
import java.security.spec.InvalidKeySpecException;
import java.security.spec.PKCS8EncodedKeySpec;
import java.security.spec.X509EncodedKeySpec;

import javax.crypto.BadPaddingException;
import javax.crypto.Cipher;
import javax.crypto.IllegalBlockSizeException;
import javax.crypto.NoSuchPaddingException;

public final class RsaCryptoAsymmetricKey {

    // *** POINT 1 *** Explicitly specify the encryption mode and the padding.
    // *** POINT 2 *** Use strong encryption methods (specifically, technologies
    // that meet the relevant criteria), including algorithms, block cipher
    // modes, and padding modes..
    // Parameters passed to getInstance method of the Cipher class: Encryption
    // algorithm, block encryption mode, padding rule.
    // In this sample, we choose the following parameter values: encryption
    // algorithm=RSA, block encryption mode=NONE, padding rule=OAEPPADDING.
```

(continues on next page)

(continued from previous page)

```
    private static final String TRANSFORMATION = "RSA/NONE/OAEPPADDING";

    // encryption algorithm
    private static final String KEY_ALGORITHM = "RSA";

    // *** POINT 3 *** Use a key of length sufficient to guarantee the strength of
    // encryption.
    // Check the length of the key
    private static final int MIN_KEY_LENGTH = 2000;

    RsaCryptoAsymmetricKey() {
    }

    public final byte[] encrypt(final byte[] plain, final byte[] keyData) {
        byte[] encrypted = null;

        try {
            // *** POINT 1 *** Explicitly specify the encryption mode and the
            // padding.
            // *** POINT 2 *** Use strong encryption methods (specifically,
            // technologies that meet the relevant criteria), including
            // algorithms, block cipher modes, and padding modes..
            Cipher cipher = Cipher.getInstance(TRANSFORMATION);

            PublicKey publicKey = generatePubKey(keyData);
            if (publicKey != null) {
                cipher.init(Cipher.ENCRYPT_MODE, publicKey);
                encrypted = cipher.doFinal(plain);
            }
        } catch (NoSuchAlgorithmException e) {
        } catch (NoSuchPaddingException e) {
        } catch (InvalidKeyException e) {
        } catch (IllegalBlockSizeException e) {
        } catch (BadPaddingException e) {
        } finally {
        }

        return encrypted;
    }

    public final byte[] decrypt(final byte[] encrypted, final byte[] keyData) {
        // In general, decryption procedures should be implemented on the server
        // side; however, in this sample code we have implemented decryption
        // processing within the application to ensure confirmation of proper
        // execution.
        // When using this sample code in real-world applications, be careful
        // not to retain any private keys within the application.

        byte[] plain = null;

        try {
            // *** POINT 1 *** Explicitly specify the encryption mode and the
            // padding.
            // *** POINT 2 *** Use strong encryption methods (specifically,
            // technologies that meet the relevant criteria), including
            // algorithms, block cipher modes, and padding modes..
```

(continues on next page)

(continued from previous page)

```
            Cipher cipher = Cipher.getInstance(TRANSFORMATION);

            PrivateKey privateKey = generatePriKey(keyData);
            cipher.init(Cipher.DECRYPT_MODE, privateKey);

            plain = cipher.doFinal(encrypted);
        } catch (NoSuchAlgorithmException e) {
        } catch (NoSuchPaddingException e) {
        } catch (InvalidKeyException e) {
        } catch (IllegalBlockSizeException e) {
        } catch (BadPaddingException e) {
        } finally {
        }

        return plain;
    }

    private static final PublicKey generatePubKey(final byte[] keyData) {
        PublicKey publicKey = null;
        KeyFactory keyFactory = null;

        try {
            keyFactory = KeyFactory.getInstance(KEY_ALGORITHM);
            publicKey = keyFactory.generatePublic(new X509EncodedKeySpec(keyData));
        } catch (IllegalArgumentException e) {
        } catch (NoSuchAlgorithmException e) {
        } catch (InvalidKeySpecException e) {
        } finally {
        }

        // *** POINT 3 *** Use a key of length sufficient to guarantee
        // the strength of encryption.
        // Check the length of the key
        if (publicKey instanceof RSAPublicKey) {
            int len = ((RSAPublicKey) publicKey).getModulus().bitLength();
            if (len < MIN_KEY_LENGTH) {
                publicKey = null;
            }
        }

        return publicKey;
    }

    private static final PrivateKey generatePriKey(final byte[] keyData) {
        PrivateKey privateKey = null;
        KeyFactory keyFactory = null;

        try {
            keyFactory = KeyFactory.getInstance(KEY_ALGORITHM);
            privateKey =
                keyFactory.generatePrivate(new PKCS8EncodedKeySpec(keyData));
        } catch (IllegalArgumentException e) {
        } catch (NoSuchAlgorithmException e) {
        } catch (InvalidKeySpecException e) {
        } finally {
        }
```

(continues on next page)

(continued from previous page)

```
        return privateKey;
    }
}
```

5.6.1.3 Encrypting and Decrypting Using Pre Shared Keys

Pre shared keys may be used when working with large data sets or to protect the confidentiality of an application's or a user's assets.

Points:

1. Explicitly specify the encryption mode and the padding
2. Use strong encryption methods (specifically, technologies that meet the relevant criteria), including algorithms, block cipher modes, and padding modes.
3. Use a key of length sufficient to guarantee the strength of encryption.

AesCryptoPreSharedKey.java

```
/*
 * Copyright (C) 2012-2026 Japan Smartphone Security Association
 *
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 *
 *      http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */
package org.jssec.android.cryptsymmetricpresharedkey;

import java.security.InvalidAlgorithmParameterException;
import java.security.InvalidKeyException;
import java.security.NoSuchAlgorithmException;

import javax.crypto.BadPaddingException;
import javax.crypto.Cipher;
import javax.crypto.IllegalBlockSizeException;
import javax.crypto.NoSuchPaddingException;
import javax.crypto.SecretKey;
import javax.crypto.spec.IvParameterSpec;
import javax.crypto.spec.SecretKeySpec;

public final class AesCryptoPreSharedKey {

    // *** POINT 1 *** Explicitly specify the encryption mode and the padding.
    // *** POINT 2 *** Use strong encryption methods (specifically, technologies
    // that meet the relevant criteria), including algorithms, block cipher
    // modes, and padding modes.
    // Parameters passed to getInstance method of the Cipher class: Encryption
    // algorithm, block encryption mode, padding rule.
```

(continues on next page)

(continued from previous page)

```
    // In this sample, we choose the following parameter values: encryption
    // algorithm=AES, block encryption mode=CBC, padding rule=PKCS7Padding
    private static final String TRANSFORMATION = "AES/CBC/PKCS7Padding";

    // Encryption algorithm
    private static final String KEY_ALGORITHM = "AES";

    // Length of IV in bytes
    public static final int IV_LENGTH_BYTES = 16;

    // *** POINT 3 *** Use a key of length sufficient to guarantee the strength of
    // encryption
    // Check the length of the key
    private static final int MIN_KEY_LENGTH_BYTES = 16;

    private byte[] mIV = null;

    public byte[] getIV() {
        return mIV;
    }

    AesCryptoPreSharedKey(final byte[] iv) {
        mIV = iv;
    }

    AesCryptoPreSharedKey() {
    }

    public final byte[] encrypt(final byte[] keyData, final byte[] plain) {
        byte[] encrypted = null;

        try {
            // *** POINT 1 *** Explicitly specify the encryption mode and the
            // padding.
            // *** POINT 2 *** Use strong encryption methods (specifically,
            // technologies that meet the relevant criteria), including
            // algorithms, block cipher modes, and padding modes.
            Cipher cipher = Cipher.getInstance(TRANSFORMATION);

            SecretKey secretKey = generateKey(keyData);
            if (secretKey != null) {
                cipher.init(Cipher.ENCRYPT_MODE, secretKey);
                mIV = cipher.getIV();

                encrypted = cipher.doFinal(plain);
            }
        } catch (NoSuchAlgorithmException e) {
        } catch (NoSuchPaddingException e) {
        } catch (InvalidKeyException e) {
        } catch (IllegalBlockSizeException e) {
        } catch (BadPaddingException e) {
        } finally {
        }

        return encrypted;
    }
```

(continues on next page)

(continued from previous page)

```
    public final byte[] decrypt(final byte[] keyData, final byte[] encrypted) {
        byte[] plain = null;

        try {
            // *** POINT 1 *** Explicitly specify the encryption mode and the
            // padding.
            // *** POINT 2 *** Use strong encryption methods (specifically,
            // technologies that meet the relevant criteria), including
            // algorithms, block cipher modes, and padding modes.
            Cipher cipher = Cipher.getInstance(TRANSFORMATION);

            SecretKey secretKey = generateKey(keyData);
            if (secretKey != null) {
                IvParameterSpec ivParameterSpec = new IvParameterSpec(mIV);
                cipher.init(Cipher.DECRYPT_MODE, secretKey, ivParameterSpec);

                plain = cipher.doFinal(encrypted);
            }
        } catch (NoSuchAlgorithmException e) {
        } catch (NoSuchPaddingException e) {
        } catch (InvalidKeyException e) {
        } catch (InvalidAlgorithmParameterException e) {
        } catch (IllegalBlockSizeException e) {
        } catch (BadPaddingException e) {
        } finally {
        }

        return plain;
    }

    private static final SecretKey generateKey(final byte[] keyData) {
        SecretKey secretKey = null;

        try {
            // *** POINT 3 *** Use a key of length sufficient to guarantee the
            // strength of encryption
            if (keyData.length >= MIN_KEY_LENGTH_BYTES) {
                // *** POINT 2 *** Use strong encryption methods (specifically,
                // technologies that meet the relevant criteria), including
                // algorithms, block cipher modes, and padding modes.
                secretKey = new SecretKeySpec(keyData, KEY_ALGORITHM);
            }
        } catch (IllegalArgumentException e) {
        } finally {
        }

        return secretKey;
    }
}
```

5.6.1.4 Using Password-based Keys to Detect Data Falsification

You may use password-based (shared-key) encryption to verify the integrity of a user's data.

Points:

1. Explicitly specify the encryption mode and the padding.
2. Use strong encryption methods (specifically, technologies that meet the relevant criteria), including algorithms, block cipher modes, and padding modes.
3. When generating a key from a password, use Salt.
4. When generating a key from a password, specify an appropriate hash iteration count.
5. Use a key of length sufficient to guarantee the MAC strength.

HmacPBEKey.java

```
/*
 * Copyright (C) 2012-2026 Japan Smartphone Security Association
 *
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 *
 *      http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */

package org.jssec.android.signsymmetricpasswordbasedkey;

import java.security.InvalidKeyException;
import java.security.NoSuchAlgorithmException;
import java.security.SecureRandom;
import java.security.spec.InvalidKeySpecException;
import java.util.Arrays;

import javax.crypto.Mac;
import javax.crypto.SecretKey;
import javax.crypto.SecretKeyFactory;
import javax.crypto.spec.PBEKeySpec;

public final class HmacPBEKey {

    // *** POINT 1 *** Explicitly specify the encryption mode and the padding.
    // *** POINT 2 *** Use strong encryption methods (specifically, technologies
    // that meet the relevant criteria),
    // including algorithms, block cipher modes, and padding modes.
    // Parameters passed to the getInstance method of the Mac class:
    // Authentication mode
    private static final String TRANSFORMATION = "PBEWITHHMACSHA1";

    // A string used to fetch an instance of the class that generates the key
    private static final String KEY_GENERATOR_MODE = "PBEWITHHMACSHA1";
```

(continues on next page)

(continued from previous page)

```
    // *** POINT 3 *** When generating a key from a password, use Salt.
    // Salt length in bytes
    public static final int SALT_LENGTH_BYTES = 20;

    // *** POINT 4 *** When generating a key from a password, specify an
    // appropriate hash iteration count.
    // Set the number of mixing repetitions used when generating keys via PBE
    private static final int KEY_GEN_ITERATION_COUNT = 1024;

    // *** POINT 5 *** Use a key of length sufficient to guarantee the MAC
    // strength.
    // Key length in bits
    private static final int KEY_LENGTH_BITS = 160;

    private byte[] mSalt = null;

    public byte[] getSalt() {
        return mSalt;
    }

    HmacPBEKey() {
        initSalt();
    }

    HmacPBEKey(final byte[] salt) {
        mSalt = salt;
    }

    private void initSalt() {
        mSalt = new byte[SALT_LENGTH_BYTES];
        SecureRandom sr = new SecureRandom();
        sr.nextBytes(mSalt);
    }

    public final byte[] sign(final byte[] plain, final char[] password) {
        return calculate(plain, password);
    }

    private final byte[] calculate(final byte[] plain, final char[] password) {
        byte[] hmac = null;

        try {
            // *** POINT 1 *** Explicitly specify the encryption mode and the
            // padding.
            // *** POINT 2 *** Use strong encryption methods (specifically,
            // technologies that meet the relevant criteria), including
            // algorithms, block cipher modes, and padding modes.
            Mac mac = Mac.getInstance(TRANSFORMATION);

            // *** POINT 3 *** When generating a key from a password, use Salt.
            SecretKey secretKey = generateKey(password, mSalt);
            mac.init(secretKey);

            hmac = mac.doFinal(plain);
        } catch (NoSuchAlgorithmException e) {
        } catch (InvalidKeyException e) {
```

(continues on next page)

(continued from previous page)

```
        } finally {
        }

        return hmac;
    }

    public final boolean verify(final byte[] hmac,
                                final byte[] plain, final char[] password) {

        byte[] hmacForPlain = calculate(plain, password);

        if (Arrays.equals(hmac, hmacForPlain)) {
            return true;
        }
        return false;
    }

    private static final SecretKey generateKey(final char[] password,
                                               final byte[] salt) {
        SecretKey secretKey = null;
        PBEKeySpec keySpec = null;

        try {
            // *** POINT 2 *** Use strong encryption methods (specifically,
            // technologies that meet the relevant criteria), including
            // algorithms, block cipher modes, and padding modes.
            // Fetch an instance of the class that generates the key
            // In this example, we use a KeyFactory that uses SHA1 to
            // generate AES-CBC 128-bit keys.
            SecretKeyFactory secretKeyFactory =
                    SecretKeyFactory.getInstance(KEY_GENERATOR_MODE);

            // *** POINT 3 *** When generating a key from a password, use Salt.
            // *** POINT 4 *** When generating a key from a password, specify an
            // appropriate hash iteration count.
            // *** POINT 5 *** Use a key of length sufficient to guarantee the MAC
            // strength.
            keySpec = new PBEKeySpec(password, salt,
                                     KEY_GEN_ITERATION_COUNT, KEY_LENGTH_BITS);
            // Clear password
            Arrays.fill(password, '?');
            // Generate the key
            secretKey = secretKeyFactory.generateSecret(keySpec);
        } catch (NoSuchAlgorithmException e) {
        } catch (InvalidKeySpecException e) {
        } finally {
            keySpec.clearPassword();
        }

        return secretKey;
    }

}
```

5.6.1.5 Using Public Keys to Detect Data Falsification

When working with data whose signature is determined using private keys stored in distinct, secure locations (such as servers), you may utilize public-key encryption for applications involving the storage of public keys on the application side solely for the purpose of authenticating data signatures.

Points:

1. Explicitly specify the encryption mode and the padding.
2. Use strong encryption methods (specifically, technologies that meet the relevant criteria), including algorithms, block cipher modes, and padding modes.
3. Use a key of length sufficient to guarantee the signature strength.

RsaSignAsymmetricKey.java

```
/*
 * Copyright (C) 2012-2026 Japan Smartphone Security Association
 *
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 *
 *      http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */

package org.jssec.android.signasymmetrickey;

import java.security.InvalidKeyException;
import java.security.KeyFactory;
import java.security.NoSuchAlgorithmException;
import java.security.PrivateKey;
import java.security.PublicKey;
import java.security.Signature;
import java.security.SignatureException;
import java.security.interfaces.RSAPublicKey;
import java.security.spec.InvalidKeySpecException;
import java.security.spec.PKCS8EncodedKeySpec;
import java.security.spec.X509EncodedKeySpec;

public final class RsaSignAsymmetricKey {

    // *** POINT 1 *** Explicitly specify the encryption mode and the padding.
    // *** POINT 2 *** Use strong encryption methods (specifically, technologies
    // that meet the relevant criteria), including algorithms, block cipher
    // modes, and padding modes.
    // Parameters passed to the getInstance method of the Cipher class:
    // Encryption algorithm, block encryption mode, padding rule.
    // In this sample, we choose the following parameter values: encryption
    // algorithm=RSA, block encryption mode=NONE, padding rule=OAEPPADDING.
    private static final String TRANSFORMATION = "SHA256withRSA";

    // encryption algorithm
```

(continues on next page)

(continued from previous page)

```
    private static final String KEY_ALGORITHM = "RSA";

    // *** POINT 3 *** Use a key of length sufficient to guarantee the signature
    // strength.
    // Check the length of the key
    private static final int MIN_KEY_LENGTH = 2000;

    RsaSignAsymmetricKey() {
    }

    public final byte[] sign(final byte[] plain, final byte[] keyData) {
        // In general, signature procedures should be implemented on the server
        // side; however, in this sample code we have implemented signature
        // processing within the application to ensure confirmation of proper
        // execution.
        // When using this sample code in real-world applications, be careful
        // not to retain any private keys within the application.

        byte[] sign = null;

        try {
            // *** POINT 1 *** Explicitly specify the encryption mode and the
            // padding.
            // *** POINT 2 *** Use strong encryption methods (specifically,
            // technologies that meet the relevant criteria), including
            // algorithms, block cipher modes, and padding modes.
            Signature signature = Signature.getInstance(TRANSFORMATION);

            PrivateKey privateKey = generatePriKey(keyData);
            signature.initSign(privateKey);
            signature.update(plain);

            sign = signature.sign();
        } catch (NoSuchAlgorithmException e) {
        } catch (InvalidKeyException e) {
        } catch (SignatureException e) {
        } finally {
        }

        return sign;
    }

    public final boolean verify(final byte[] sign,
                                final byte[] plain, final byte[] keyData) {

        boolean ret = false;

        try {
            // *** POINT 1 *** Explicitly specify the encryption mode and the
            // padding.
            // *** POINT 2 *** Use strong encryption methods (specifically,
            // technologies that meet the relevant criteria), including
            // algorithms, block cipher modes, and padding modes.
            Signature signature = Signature.getInstance(TRANSFORMATION);

            PublicKey publicKey = generatePubKey(keyData);
```

(continues on next page)

(continued from previous page)

```
            signature.initVerify(publicKey);
            signature.update(plain);

            ret = signature.verify(sign);

        } catch (NoSuchAlgorithmException e) {
        } catch (InvalidKeyException e) {
        } catch (SignatureException e) {
        } finally {
        }

        return ret;
    }

    private static final PublicKey generatePubKey(final byte[] keyData) {
        PublicKey publicKey = null;
        KeyFactory keyFactory = null;

        try {
            keyFactory = KeyFactory.getInstance(KEY_ALGORITHM);
            publicKey = keyFactory.generatePublic(new X509EncodedKeySpec(keyData));
↪
        } catch (IllegalArgumentException e) {
        } catch (NoSuchAlgorithmException e) {
        } catch (InvalidKeySpecException e) {
        } finally {
        }

        // *** POINT 3 *** Use a key of length sufficient to guarantee the
        // signature strength.
        // Check the length of the key
        if (publicKey instanceof RSAPublicKey) {
            int len = ((RSAPublicKey) publicKey).getModulus().bitLength();
            if (len < MIN_KEY_LENGTH) {
                publicKey = null;
            }
        }

        return publicKey;
    }

    private static final PrivateKey generatePriKey(final byte[] keyData) {
        PrivateKey privateKey = null;
        KeyFactory keyFactory = null;

        try {
            keyFactory = KeyFactory.getInstance(KEY_ALGORITHM);
            privateKey = keyFactory
                    .generatePrivate(new PKCS8EncodedKeySpec(keyData));
        } catch (IllegalArgumentException e) {
        } catch (NoSuchAlgorithmException e) {
        } catch (InvalidKeySpecException e) {
        } finally {
        }

        return privateKey;
```

(continues on next page)

(continued from previous page)

```
    }
}
```

5.6.1.6 Using Pre Shared Keys to Detect Data Falsification

You may use pre-shared keys to verify the integrity of application assets or user assets.

Points:

1. Explicitly specify the encryption mode and the padding.
2. Use strong encryption methods (specifically, technologies that meet the relevant criteria), including algorithms, block cipher modes, and padding modes.
3. Use a key of length sufficient to guarantee the MAC strength.

HmacPreSharedKey.java

```
/*
 * Copyright (C) 2012-2026 Japan Smartphone Security Association
 *
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 *
 *      http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */

package org.jssec.android.signsymmetricpresharedkey;

import java.security.InvalidKeyException;
import java.security.NoSuchAlgorithmException;
import java.util.Arrays;

import javax.crypto.Mac;
import javax.crypto.SecretKey;
import javax.crypto.spec.SecretKeySpec;

public final class HmacPreSharedKey {

    // *** POINT 1 *** Explicitly specify the encryption mode and the padding.
    // *** POINT 2 *** Use strong encryption methods (specifically, technologies
    // that meet the relevant criteria), including algorithms, block cipher
    // modes, and padding modes.
    // Parameters passed to the getInstance method of the Mac class:
    // Authentication mode
    private static final String TRANSFORMATION = "HmacSHA256";

    // Encryption algorithm
    private static final String KEY_ALGORITHM = "HmacSHA256";

    // *** POINT 3 *** Use a key of length sufficient to guarantee the MAC
```

(continues on next page)

(continued from previous page)

```
    // strength.
    // Check the length of the key
    private static final int MIN_KEY_LENGTH_BYTES = 16;

    HmacPreSharedKey() {
    }

    public final byte[] sign(final byte[] plain, final byte[] keyData) {
        return calculate(plain, keyData);
    }

    public final byte[] calculate(final byte[] plain, final byte[] keyData) {
        byte[] hmac = null;

        try {
            // *** POINT 1 *** Explicitly specify the encryption mode and the
            // padding.
            // *** POINT 2 *** Use strong encryption methods (specifically,
            // technologies that meet the relevant criteria), including
            // algorithms, block cipher modes, and padding modes.
            Mac mac = Mac.getInstance(TRANSFORMATION);

            SecretKey secretKey = generateKey(keyData);
            if (secretKey != null) {
                mac.init(secretKey);

                hmac = mac.doFinal(plain);
            }
        } catch (NoSuchAlgorithmException e) {
        } catch (InvalidKeyException e) {
        } finally {
        }

        return hmac;
    }

    public final boolean verify(final byte[] hmac,
                                final byte[] plain, final byte[] keyData) {
        byte[] hmacForPlain = calculate(plain, keyData);

        if (hmacForPlain != null && Arrays.equals(hmac, hmacForPlain)) {
            return true;
        }

        return false;
    }

    private static final SecretKey generateKey(final byte[] keyData) {
        SecretKey secretKey = null;

        try {
            // *** POINT 3 *** Use a key of length sufficient to guarantee the MAC
            // strength.
            if (keyData.length >= MIN_KEY_LENGTH_BYTES) {
                // *** POINT 2 *** Use strong encryption methods (specifically,
                // technologies that meet the relevant criteria), including
```

(continues on next page)

(continued from previous page)

```
                // algorithms, block cipher modes, and padding modes.
                secretKey = new SecretKeySpec(keyData, KEY_ALGORITHM);
            }
        } catch (IllegalArgumentException e) {
        } finally {
        }

        return secretKey;
    }
}
```

5.6.2 Rule Book

When using encryption technology, it is important to obey the following rules.

1. *When Specifying an Encryption Algorithm, Explicitly Specify the Encryption Mode and the Padding (Required)*
2. *Use Strong Algorithms (Specifically, Algorithms that Meet the Relevant Criteria) (Required)*
3. *When Using Password-based Encryption, Do Not Store Passwords on Device (Required)*
4. *When Generating Keys from Passwords, Use Salt (Required)*
5. *When Generating Key from Password, Specify Appropriate Hash Iteration Count (Required)*
6. *Take Steps to Increase the Strengths of Passwords (Recommended)*

5.6.2.1 When Specifying an Encryption Algorithm, Explicitly Specify the Encryption Mode and the Padding (Required)

When using cryptographic technologies such as encryption and data verification, it is important that the encryption mode and the padding be explicitly specified. When using encryption in Android application development, you will primarily use the Cipher class within java.crypto. To use the Cipher class, you will first create an instance of Cipher class object by specifying the type of encryption to use. This specification is called a Transformation, and there are two formats in which Transformations may be specified:

- “algorithm/mode/padding”
- “algorithm”

In the latter case, the encryption mode and the padding will be implicitly set to the appropriate default values for the encryption service provider that Android may access. These default values are chosen to prioritize convenience and compatibility and in some cases may not be particularly secure choices. For this reason, to ensure proper security protections it is mandatory to use the former of the two formats, in which the encryption mode and padding are explicitly specified.

5.6.2.2 Use Strong Algorithms (Specifically, Algorithms that Meet the Relevant Criteria) (Required)

When using cryptographic technologies it is important to choose strong algorithms which meet certain criteria. In addition, in cases where an algorithm allows multiple key lengths, it is important to consider the application’s full product lifetime and to choose keys of length sufficient to guarantee security. Moreover, for some encryption modes and padding modes there exist known strategies of attack; it is important to make choices that are robust against such threats.

Indeed, choosing weak encryption methods can have disastrous consequences; for example, files which were supposedly encrypted to prevent eavesdropping by a third party may in fact be only ineffectually protected and may allow

third-party eavesdropping. Because the continual progress of IT leads to continual improvements in encryption-analysis technologies, it is crucial to consider and select algorithms that can guarantee security throughout the entire period during which you expect an application to remain in operation.

Algorithm Security Lifetime, that are expected to be secure for the entire security life of the protected data and Standards for actual encryption technologies differ from country to country, as detailed in the tables below.

Table 5.6.1: NIST, FIPS(USA)[26][27][28]

Algorithm Security Lifetimes	Symmetric Key algorithms	Asymmetric Key algorithms(e.g., RSA, DSA, DH)	Elliptic-curve cryptography(e.g., ECDSA)	Digital Signatures and hash-only applications	HMAC,Key Derivation Functions, Random Number Generation
Legacy(max. of 80 bits of strength)	80	1024	160	160	160
Through 2030(min. of 112 bits of strength)	112	2048	224	224	160
Beyond 2030(min. of 128 bits of strength)	128	3072	256	256	160

Unit bit

Table 5.6.2: ECRYPT II (EU)[29]

Algorithm Security Lifetimes	Symmetric Key algorithms	Asymmetric Key algorithms	Elliptic-curve cryptography	HASH
less or equal to 4 years protection	80	1248	160	160
10 years protection	96	1776	192	192
20 years protection	112	2432	224	224
30 years protection	128	3248	256	256
Good protection against quantum computers unless Shor's algorithm applies.	256	15424	512	512

Unit bit

[26] NIST Special Publication 800-57 Part1 Revision4(1/28/2016) "Recommendation for Key Management Part1:General" "5.6 Guidance for Cryptographic Algorithm and Key Size Selection" (https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-57pt1r4.pdf)

[27] NIST Special Publication 800-131A Revision2(3/21/2019) "Transitioning the Use of Cryptographic Algorithms and Key Lengths" "1.1 Background and Purpose" "1.2.1 Security Strengths" (https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-131Ar2.pdf)

[28] Implementation Guidance for FIPS 140-2 and the Cryptographic Module Validation Program(Last Modified Date:05/10/2017) "7.5 Strength of Key Establishment Methods" (https://csrc.nist.gov/csrc/media/projects/cryptographic-module-validation-program/documents/fips140-2/fips1402ig.pdf)

[29] "ECRYPT II Yearly Report on Algorithms and Keysizes(2011-2012)" (European Network of Excellence for Cryptology II, Revision 1.0, 30. Sept 2012 (http://www.ecrypt.eu.org/ecrypt2/documents/D.SPA.20.pdf)

Table 5.6.3: CRYPTREC(Japan) CRYPTREC Ciphers List[30]

Technology family		Name
Public-key cryptography	Signature	DSA, ECDSA, RSA=PSS, RSASSA=PKCS1=V1_5
	Confidentiality	RSA-OAEP
	Key sharing	DH, ECDH
Shared-key cryptography	64 bit block encryption	3-key Triple DES
	128 bit block encryption	AES, Camellia
	Stream encryption	KCipher-2
Hash function		SHA-256, SHA-384, SHA-512
Encryption usage mode	Cipher mode	CBC, CFB, CTR, OFB
	Authenticated cipher modes	CCM, GCM
Message authentication codes		CMAC, HMAC
Entity authentication		ISO/IEC 9798-2, ISO/IEC 9798-3

5.6.2.3 When Using Password-based Encryption, Do Not Store Passwords on Device (Required)

In password-based encryption, when generating an encryption key based on a password input by a user, do not store the password within the device. The advantage of password-based encryption is that it eliminates the need to manage encryption keys; storing the password on the device eliminates this advantage. Needless to say, storing passwords on a device invites the risk of eavesdropping by other applications, and thus storing passwords on devices is also unacceptable for security reasons.

5.6.2.4 When Generating Keys from Passwords, Use Salt (Required)

In password-based encryption, when generating an encryption key based on a password input by a user, always use Salt. In addition, if you are providing features to different users within the same device, use a different Salt for each user. The reason for this is that, if you generate encryption keys using only a simple hash function without using Salt, the passwords may be easily recovered using a technique known as a "rainbow table." When Salt is applied, keys generated from the same password will be distinct (different hash values), preventing the use of a rainbow table to search for keys.

(Sample) When generating keys from passwords, use salt

```
public final byte[] encrypt(final byte[] plain, final char[] password) {
    byte[] encrypted = null;

    try {
        // *** POINT *** Explicitly specify the encryption mode
        // and the padding.

        // *** POINT *** Use strong encryption methods (specifically,
        // technologies that meet the relevant criteria),
        // including algorithms, block cipher modes, and padding modes.
        Cipher cipher = Cipher.getInstance(TRANSFORMATION);

        // *** POINT *** When generating keys from passwords, use Salt.
        SecretKey secretKey = generateKey(password, mSalt);
```

[30] https://www.cryptrec.go.jp/list.html

5.6.2.5 When Generating Key from Password, Specify Appropriate Hash Iteration Count (Required)

In password-based encryption, when generating an encryption key based on a password input by a user, you will choose a number of times for the hashing procedure to be repeated during the process of key generation ("stretching"); it is important to specify this number large enough to ensure security. In general, the iteration count equal to 1,000 or greater is considered sufficient. If you are using the key to protect even more valuable assets, specify a count equal to 1,000,000 or greater. Because the processing time required for a single computation of the hash function is minuscule, it may be easy for attackers to launch brute-force attacks. Thus, by using the stretching method - in which hash processing is repeated many times - we can purposely ensure that the process consumes significant time and thus that brute-force attacks are more costly. Note that the number of stretching repetitions will also affect your application's processing speed, so take care in choosing an appropriate value.

(Sample) When generating key from password, Set hash iteration counts

```
private static final SecretKey generateKey(final char[] password,
                                           final byte[] salt) {
    SecretKey secretKey = null;
    PBEKeySpec keySpec = null;

    (Omit)

    // *** POINT *** When generating a key from password, use Salt.
    // *** POINT *** When generating a key from password, specify
    // an appropriate hash iteration count.
    // ** POINT *** Use a key of length sufficient to guarantee
    // the strength of encryption.
    keySpec = new PBEKeySpec(password,
                             salt,
                             KEY_GEN_ITERATION_COUNT,
                             KEY_LENGTH_BITS);
```

5.6.2.6 Take Steps to Increase the Strengths of Passwords (Recommended)

In password-based encryption, when generating an encryption key based on a password input by a user, the strength of the generated key is strongly affected by the strength of the user's password, and thus it is desirable to take steps to strengthen the passwords received from users. For example, you might require that passwords be at least 8 characters long and contain multiple types of characters—perhaps at least one letter, one numeral, and one symbol.

5.6.3 Advanced Topics

5.6.3.1 Choosing encryption methods

In the above "sample codes", we showed implementation examples involving three types of cryptographic methods each for encryption and decryption and for detecting data falsification. You may use "Fig. 5.6.1 *Selection flowchart for sample code to protect data from eavesdropping*", "Fig. 5.6.2 *Selection flowchart for sample code to detect falsifications*" to make a coarse-grained choice of which cryptographic method to use based on your application. On the other hand, more fine-tuned choices of cryptographic methods require more detailed comparisons of the features of various methods. In what follows we consider some of these comparisons.

Comparison of cryptographic methods for encryption and decryption

Public-key cryptography has high processing cost and thus is not well suited for large-scale data processing. However, because the keys used for encryption and for decryption are different, it is relatively easy to manage keys in cases where you handle only the public key on the application side (i.e. you only perform encryption) and perform decryption in a separate (secure) location. Shared-key cryptography is an all-purpose encryption scheme with few limitations, but in this case the same key is used for encryption and decryption, and thus it is necessary to store the key securely

within the application, making key management difficult. Password-based cryptography (shared-key cryptography based on a password) generates keys from user-specified passwords, obviating the need to store key-related secrets within devices. This method is used for applications protecting only user assets but not application assets. Because the strength of the encryption depends on the strength of the password, it is necessary to choose passwords whose complexity grows in proportion to the value of assets to be protected. Please refer to "5.6.2.6. *Take Steps to Increase the Strengths of Passwords (Recommended)*".

Table 5.6.4: Comparison of cryptographic methods for encryption and decryption

	Public key	**Shared key**	**Password-based**
Processing of large-scale data	NO (processing cost too high)	OK	OK
Protecting application (or service) assets	OK	OK	NO (allows eavesdropping by users)
Protecting user assets	OK	OK	OK
Strength of encryption	Depends on key length	Depends on key length	Depends on strength of password, on Salt, and on the number of hash repetitions
Key storage	Easy (only public keys)	Difficult	Easy
Processing carried out by application	Encryption (decryption is done on servers or elsewhere)	Encryption and decryption	Encryption and decryption

Comparison of cryptographic methods for detecting data falsification

The comparison here is similar to that discussed above for encryption and decryption, with the exception that that table item corresponding to data size is no longer relevant.

Table 5.6.5: Comparison of cryptographic methods for detecting data falsification

	Public key	**Shared key**	**Password-based**
Protecting application (or service) assets	OK	OK	NO (allows falsification by users)
Protecting user assets	OK	OK	OK
Strength of encryption	Depends on key length	Depends on key length	Depends on strength of password, on Salt, and on the number of hash repetitions
Key storage	Easy (only public keys)	Difficult(Refer to "5.6.3.3. *Protecting Key*")	Easy
Processing carried out by application	Encryption (decryption is done on servers or elsewhere)	MAC computation, MAC verification	MAC computation, MAC verification

MAC: Message authentication code

Note that these guidelines are primarily concerned with the protection of assets deemed low-level or medium-level assets according to the classification discussed in Section "3.1.3. *Asset Classification and Protective Countermeasures*". Because the use of encryption involves the consideration of a greater number of issues—such as the problem of key storage—than other preventative measures (such as access controls), encryption should only be considered for cases in which assets cannot be adequately protected within the Android OS security mode.

5.6.3.2 Generation of random numbers

When using cryptographic technologies, it is extremely important to choose strong encryption algorithms and encryption modes and sufficiently long keys in order to ensure the security of the data handled by applications and services. However, even if all of these choices are made appropriately, the strength of the security guaranteed by the algorithms in use plummets immediately to zero when the keys that form the linchpin of the security protocol are leaked or guessed. Even for the initial vector (IV) used for shared-key encryption under AES and similar protocols, or the Salt used for password-based encryption, large biases can make it easy for third parties to launch attacks, heightening the risk of exposure to data leakage or corruption. To prevent such situations, it is necessary to generate keys and IVs in such a way as to make it difficult for third parties to guess their values, and random numbers play an immensely important role in ensuring the realization of this imperative. A device that generates random numbers is called a random-number generator. Whereas hardware random-number generators (RNGs) may use sensors or other devices to produce random numbers by measuring natural phenomena that cannot be predicted or reproduced, it is more common to encounter software-implemented random-number generators, known as pseudorandom-number generators (PRNGS).

In Android applications, random numbers of sufficient security for use in encryption may be generated via the SecureRandom class. The SecureRandom class can internally have multiple implementations, which are called providers and provide the function, and if no provider is explicitly specified, then the default provider will be selected. Crypto Provider, which provides the SHA1PRNG algorithm that is cryptographically unsafe[31], was deprecated in Android 7.0 (API level 24), and it was removed in Android 9.0 (API level 28)[32][33][34]. If Crypto Provider is specified and SecureRandom is used, NoSuchProviderException will always occur in devices running Android 9.0 and higher, and NoSuchProviderException will occur even in devices running Android 7.0 and higher if targetSdkVersion>=24. For this reason, generally, the use of SecureRandom without specifying the provider is recommended. In what follows we offer examples to demonstrate the use of SecureRandom.

Using SecureRandom (using the default implementation)

```
import java.security.SecureRandom;
[...]
    SecureRandom random = new SecureRandom();
    byte[] randomBuf = new byte [128];

    random.nextBytes(randomBuf);
[...]
```

The pseudorandom-number generators found in programs like SecureRandom typically operate on the basis of a process like that illustrated in "Fig. 5.6.3 *Inner process of pseudorandom number generator*". A random number seed is entered to initialize the internal state; thereafter, the internal state is updated each time a random number is generated, allowing the generation of a sequence of random numbers.

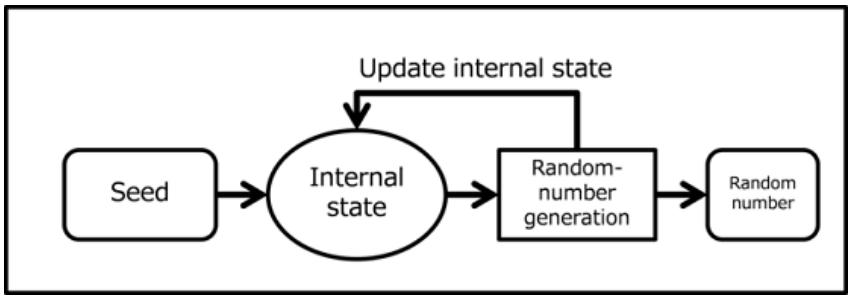


Fig. 5.6.3: Inner process of pseudorandom number generator

[31] On statistical distance based testing of pseudo random sequences and experiments with PHP and Debian OpenSSL - 8.1 Java SHA1PRNG API based sequences (https://webpages.uncc.edu/yonwang/papers/lilesorics.pdf)

[32] Security "Crypto" provider deprecated in Android N (https://android-developers.googleblog.com/2016/06/security-crypto-provider-deprecated-in.html)

[33] Cryptography Changes in Android P (https://android-developers.googleblog.com/2018/03/cryptography-changes-in-android-p.html)

[34] SecureRandom (https://developer.android.com/reference/java/security/SecureRandom)

Random number seeds

The seed plays an extremely important role in a pseudorandom number generator (PRNG).

As noted above, PRNGs must be initialized by specifying a seed. Thereafter, the process used to generate random numbers is a deterministic algorithm, so if you specify the same seed you will get the same sequence of random numbers. This means that if a third party gains access to (that is, eavesdrops upon) or guesses the seed of a PRNG, he can produce the same sequence of random numbers, thus destroying the properties of confidentiality and integrity that the random numbers provide.

For this reason, the seed of a random number generator is itself a highly confidential piece of information—and one which must be chosen in such a way as to be impossible to predict or guess. For example, time information or device-specific data (such as a MAC address, IMEI, or Android ID) should not be used to construct RNG seeds. On many Android devices, /dev/urandom or /dev/random is available, and the default implementation of SecureRandom provided by Android uses these device files to determine seeds for random number generators. As far as confidentiality is concerned, as long as the RNG seed exists only in memory, there is little risk of discovery by third parties with the exception of malware tools that acquire root privileges. If you need to implement security measures that remain effective even on rooted devices, consult an expert in secure design and implementation.

The internal state of a pseudorandom number generator

The internal state of a pseudorandom number generator is initialized by the seed, then updated each time a random number is generated. Just as for the case of PRNGs initialized by the same seed, two PRNGs with the same internal state will subsequently produce precisely the same sequence of random numbers. Consequently, it is also important to protect the internal state against eavesdropping by third parties. However, because the internal state exists in memory, there is little risk of discovery by third parties except in cases involving malware tools that acquire root access. If you need to implement security measures that remain effective even on rooted devices, consult an expert in secure design and implementation.

5.6.3.3 Protecting Key

When using encryption techniques to ensure the security (confidentiality and integrity) of sensitive data, even the most robust encryption algorithm and key lengths will not protect data from third-party attacks if the data content of the keys themselves are readily available. For this reason, the proper handling of keys is among the most important items to consider when using encryption. Of course, depending on the level of the assets you are attempting to protect, the proper handling of keys may require extremely sophisticated design and implementation techniques which exceed the scope of these guidelines. Here we can only offer some basic ideas regarding the secure handling of keys for various applications and key storage locations; our discussion does not extend to specific implementation methods, and as necessary we recommend that you consult an expert in secure design and implementation for Android.

To begin, "Fig. 5.6.4 *Places of encrypt keys and strategies for protecting them*" illustrates the various places in which keys used for encryption and related purposes in Android smartphones and tablets may exist, and outlines strategies for protecting them.

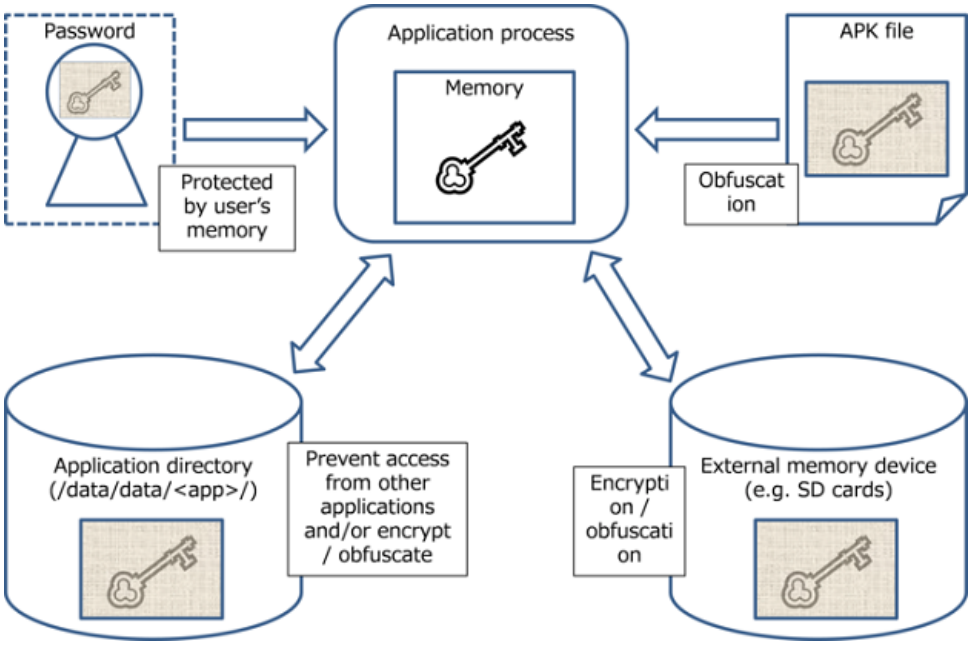


Fig. 5.6.4: Places of encrypt keys and strategies for protecting them

The table below summarizes the asset classes of the assets protected by keys, as well as the protection policies appropriate for various asset owners. For more information on asset classes, please refer to "3.1.3. *Asset Classification and Protective Countermeasures*".

Table 5.6.6: Asset classification and protective countermeasures-1

Asset owner	**Device User**		**Application / Service Provider**	
Asset level	**High**	**Medium / Low**	**High**	**Medium / Low**
Key storage location	**Protection policy**			
User's memory	Improve password strength		Disallow the use of user passwords	
Application directory (non-public storage)	Encryption or obfuscation of key data	Forbid read/write operations from outside the application	Encryption or obfuscation of key data	Forbid read/write operations from outside the application

If keys are stored in public strage such as an APK file or an SD card, it is as follows.

Table 5.6.7: Asset classification and protective countermeasures-2

Key storage location	**Protection policy**
APK file	Obfuscation of key data Note:Be aware that most Java obfuscation tools, such as Proguard, do not obfuscate data (character) strings.
SD card or elsewhere (public storage)	Encryption or obfuscation of key data

In what follows, we will augment the discussion of protective measures appropriate for the various places in which keys may be stored.

Keys stored in a user's memory

Here we are considering password-based encryption. When keys are generated from passwords, the key storage location is the user's memory, so there is no danger of leakage due to malware. However, depending on the strength of the password, it may be easy to reproduce keys. For this reason, it is necessary to take steps—similar to those taken when asking users to specify service login passwords—to ensure the strength of passwords; for example, passwords may be restricted by the UI, or warning messages may be used. Please refer to "5.5.2.6. *Place a summary version*

of the application privacy policy in the assets folder (Recommended)". Of course, when passwords are stored in a user's memory one must keep in mind the possibility that the password will be forgotten. To ensure that data may be recovered in the event of a forgotten password, it is necessary to store backup data in a secure location other than the device (for example, on a server).

Keys stored in application directories

When keys are stored in Private mode in application directories, the key data cannot be read by other applications. In addition, if the application has disabled backup functionality, users will also be unable to access the data. Thus, when storing keys used to protect application assets in application directories, you should disable backups.

However, if you also need to protect keys from applications or users with root privileges, you must encrypt or obfuscate the keys. For keys used to protect user assets, you may use password-based encryption. For keys used to encrypt application assets that you wish to keep private from users as well, you must store the key used for key encryption in an APK file, and the key data must be obfuscated.

Keys stored in APK Files

Because data in APK files may be accessed, in general this is not an appropriate place to store confidential data such as keys. When storing keys in APK files, you must obfuscate the key data and take steps to ensure that the data may not be easily read from the APK file.

Keys stored in public storage locations (such as SD cards)

Because public storage can be accessed by all applications, in general it is not an appropriate place to store confidential data such as passwords. When storing keys in public locations, it is necessary to encrypt or obfuscate the key data to ensure that the data cannot be easily accessed. See also the protections suggested above under "Keys stored in application directories" for cases in which keys must also be protected from applications or users with root privileges.

Handling of keys within process memory

When using the cryptographic technologies available in Android, key data that have been encrypted or obfuscated somewhere other than the application process shown in the figure above must be decrypted (or, for password-based keys, generated) in advance of the encryption procedure; in this case, key data will reside in process memory in unencrypted form. On the other hand, the memory of an application process may not generally be read by other applications, so if the asset class falls within the range covered by these guidelines there is no particular need to take specific steps to ensure security. In cases where—due to the specific objective in question or to the level of the assets handled by an application—it is unacceptable for key data to appear in unencrypted form (even though they are present that way in process memory), it may be necessary to resort to obfuscation or other techniques for key data and encryption logic. However, these methods are difficult to realize at the Java level; instead, you will use obfuscation tools at the JNI level. Such measures fall outside the scope of these guidelines; consult an expert in secure design and implementation.

5.6.3.4 Addressing Vulnerabilities with Security Provider from Google Play Services

Google Play Services (Version 5.0 and later) provides a framework known as *Provider Installer* that may be used to address vulnerabilities in Security Provider.

First, Security Provider provides implementations of various encryption-related algorithms based on Java Cryptography Architecture (JCA). These Security Provider algorithms may be used via classes such as Cipher, Signature, and Mac to make use of encryption technology in Android apps. In general, rapid response is required whenever vulnerabilities are discovered in encryption-technology-related implementations. Indeed, the exploitation of such vulnerabilities for malicious purposes could result in major damage. Because encryption technologies are also relevant for Security Provider, it is desirable that revisions designed to address vulnerabilities be reflected as quickly as possible.

The most common method of reflecting Security Provider revisions is to use device updates. The process of reflecting revisions via device updates begins with the device manufacturer preparing an update, after which users apply this update to their devices. Thus, the question of whether or not an app has access to an up-to-date version of Security Provider—including the most recent revisions—depends in practice on compliance from both manufacturers and users. In contrast, using Provider Installer from Google Play Services ensures that apps have access to automatically-updated versions of Security Provider.

With Provider Installer from Google Play Services, calling Provider Installer from an app allows access to Security Provider as provided by Google Play Services. Google Play Services is automatically updated via the Google Play Store, and thus the Security Provider provided by Provider Installer will be automatically updated to the latest version, with no dependence on compliance from manufacturers or users.

Sample code that calls Provider Installer is shown below.

Call Provider Installer

```
import com.google.android.gms.common.GooglePlayServicesUtil;
import com.google.android.gms.security.ProviderInstaller;

public class MainActivity extends Activity
        implements ProviderInstaller.ProviderInstallListener {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        ProviderInstaller.installIfNeededAsync(this, this);
        setContentView(R.layout.activity_main);
    }

    @Override
    public void onProviderInstalled() {
        // Called when Security Provider is the latest version,
        // or when installation completes.
    }

    @Override
    public void onProviderInstallFailed(int errorCode, Intent recoveryIntent) {
        GoogleApiAvailability.getInstance().showErrorNotification(this, errorCode);
    }
}
```

5.6.3.5 Conscrypt Module

The Conscrypt module is an APEX file that is used to correct vulnerabilities that become clear through implementation of technologies related to cryptography without relying on OTA updates prepared by manufacturers.

Android specific public API for Conscrypt is not included on Android 9. However, a small number of public API methods are added in android.net.ssl on Android 10, enabling access to the Conscrypt function that is not exposed by the classes under javax.net.ssl.

The Conscrypt module uses the native library BoringSSL that was forked by Google from OpenSSL, and is applied to encryption and TLS on many Google products.

Originally, clearly requesting specific providers as shown below was not recommended. And BouncyCastle provider implementations of encryption algorithms were deleted from Android 12[35].

```
Cipher.getInstance("AES/CBC/PKCS7PADDING", "BC");
// OR
Cipher.getInstance("AES/CBC/PKCS7PADDING", Security.getProvider("BC"));
```

Applications that are affected by this change include the following.

1. Application that uses invalid key sizes with KeyGenerator

[35] At the time of this writing, it has been confirmed that no warning occurs during building and this can be run without any problems on the Android 12 emulator. However, there is no mistake that this is not recommended.

2. Application that has initialized the Galois/Counter Mode (GCM) encryption using a size other than 12 bytes for the initial vector byte size

Concerning 1, the key sizes supported by Conscrypt are 128 bits, 192 bits, and 256 bits. If a key size other than these is specified, an exception occurs while the KeyGenerator.init() method is running. In this case, it is necessary to correct the key size to an appropriate supported size.

```
keygen.init(512, random); // // Exception occurs. Caused by: java.security.
↪InvalidParameterException: Key size must be either 128, 192, or 256 bits
Concerning 2, if using a GCM encryption, the byte size for the initial vector must␣
↪be 12 bytes. If other size, for example 16 bytes, is specified, an exception␣
↪occurs when running the Cipher.init() method. In this case, it is necessary to␣
↪correct the initial vector byte size to an appropriate supported size.
```

```
SecureRandom random = SecureRandom.getInstance("SHA1PRNG");
byte[] INITIALV = new byte[16]; // // Specify initial vector byte size to 16
random.nextBytes(INITIALV);
~
byte[] iv = INITIALV;
SecretKeySpec skey = new SecretKeySpec(key, CIPHER);
IvParameterSpec ivp = new IvParameterSpec(iv);
Cipher cipher = Cipher.getInstance("AES_256/GCM/NOPADDING"); // // GCM encryption␣
↪algorithm
cipher.init(Cipher.ENCRYPT_MODE, skey, ivp); // Exception occurs. java.security.
↪InvalidAlgorithmParameterException: Expected IV length of 12 but was 16
```

5.6.3.6 Countermeasures against backup data leaks

If an attacker gains access to backup files, unencrypted data backed up by the app may be retrieved. To prevent backups when unnecessary, add this to your manifest:

```
<application android:name="com.example.foo" android:allowBackup="false">
...
</application>
```

In Android 12 and later, setting allowBackup to false restricts cloud and ADB backups but allows device-to-device migration.

If backup is required as a specification of the application, specify the type of data to be backed up as follows (Android 12 or later).

```
<application android:name="com.example.foo"
  android:dataExtractionRules="backup_rules.xml">
 --
</application>
```

```
<application android:name="com.example.foo"
  android:fullBackupContent="@xml/backup_rules">
 --
</application>
```

For more information, see the Android Developer Guide:[36]

[36] https://developer.android.com/privacy-and-security/risks/backup-leaks?hl=ja

5.6.3.7 Hard-coded Cryptographic Secrets

Developers can use cryptography to protect data confidentiality and integrity, but often do not properly leverage key storage and instead hardcode cryptographic secrets into application code or asset files, which poses serious security issues.

Hardcoded credentials and cryptographic secrets can be easily obtained using reverse engineering tools, which allows attackers to access sensitive data and creates significant security risks.

Therefore, when system-level authentication information is required, it is recommended to use the KeyChain API, and to store app-specific authentication information using the Android Keystore provider. In particular, from Android 16 onwards, StrongBox support is becoming mandatory for devices equipped with Secure Elements, and it is expected that storing credentials in KeyChain or KeyStore will become a common practice in the future. By utilizing these features, the risks associated with hardcoding can be effectively reduced. For more information, see the Android Developer Guide.[37]

Implementation Procedure

1. Use Android Keystore provider to store app-specific authentication information

Generate and store AES symmetric keys using Android Keystore.

```
companion object {
    private const val ANDROID_KEY_STORE_PROVIDER = "AndroidKeyStore"
    private const val ANDROID_KEY_STORE_ALIAS = "AES_KEY_DEMO"
}

@Throws(
    KeyStoreException::class,
    NoSuchAlgorithmException::class,
    NoSuchProviderException::class,
    InvalidAlgorithmParameterException::class
)
private fun createAndStoreSecretKey() {
    val builder: KeyGenParameterSpec.Builder = KeyGenParameterSpec.Builder(
        ANDROID_KEY_STORE_ALIAS,
        KeyProperties.PURPOSE_ENCRYPT or KeyProperties.PURPOSE_DECRYPT
    )
    val keySpec: KeyGenParameterSpec = builder
        .setKeySize(256)
        .setBlockModes(KeyProperties.BLOCK_MODE_GCM)
        .setEncryptionPaddings(KeyProperties.ENCRYPTION_PADDING_NONE)
        .setRandomizedEncryptionRequired(true)
        .build()
    val aesKeyGenerator: KeyGenerator =
        KeyGenerator.getInstance(KeyProperties.KEY_ALGORITHM_AES, ANDROID_KEY_
↪STORE_PROVIDER)
    aesKeyGenerator.init(keySpec)
    aesKeyGenerator.generateKey()
}

@Throws(
    KeyStoreException::class,
    UnrecoverableEntryException::class,
    NoSuchAlgorithmException::class,
    CertificateException::class,
    IOException::class,
    NoSuchPaddingException::class,
```

(continues on next page)

[37] https://developer.android.com/privacy-and-security/risks/hardcoded-cryptographic-secrets

(continued from previous page)

```
    InvalidKeyException::class,
    IllegalBlockSizeException::class,
    BadPaddingException::class
)
private fun encryptWithKeyStore(plainText: String): ByteArray? {
    // Initialize KeyStore
    val keyStore: KeyStore = KeyStore.getInstance(ANDROID_KEY_STORE_PROVIDER)
    keyStore.load(null)
    // Retrieve the key with alias created before
    val keyEntry: KeyStore.SecretKeyEntry =
        keyStore.getEntry(ANDROID_KEY_STORE_ALIAS, null) as KeyStore.SecretKeyEntry
    val key: SecretKey = keyEntry.secretKey
    // Use the secret key at your convenience
    val cipher: Cipher = Cipher.getInstance("AES/GCM/NoPadding")
    cipher.init(Cipher.ENCRYPT_MODE, key)
    return cipher.doFinal(plainText.toByteArray())
}
```

2. Use the KeyChain API to manage system-level credentials

Use the KeyChain API to manage certificates and private keys shared across the system

```
private fun selectCertificate() {
    KeyChain.choosePrivateKeyAlias(this, object : KeyChainAliasCallback {
        override fun alias(alias: String?) {
            if (alias != null) {
                // Use the alias to retrieve the certificate
                try {
                    val privateKey = KeyChain.getPrivateKey(this@MainActivity,␣
↪alias)
                    val certificateChain = KeyChain.
↪getCertificateChain(this@MainActivity, alias)
                    println("Selected Certificate Alias: $alias")

                    // Example usage of the private key and certificate chain
                    // Typically, you would use these for SSL/TLS connections or␣
↪signing operations
                } catch (e: KeyChainException) {
                    e.printStackTrace()
                } catch (e: InterruptedException) {
                    e.printStackTrace()
                }
            }
        }
    }, null, null, null, -1, null)
}
```

3. Obtaining and encrypting credentials, selecting a certificate

Obtain an API key from a remote server, encrypt it, and then select a system-level certificate to use.

```
try {
    createAndStoreSecretKey()
    val apiKey = fetchApiKeyFromServer()
    val encryptedData = encryptWithKeyStore(apiKey)
    // Do something with the encrypted data (e.g., log output)
    println("Encrypted API Key: ${encryptedData?.joinToString(", ")}")
```

(continues on next page)

(continued from previous page)

```
    // Use KeyChain to select and use a system-level certificate
    selectCertificate()
} catch (e: Exception) {
    e.printStackTrace()
}
```

The increasing adoption of StrongBox-equipped devices is expected to create synergies with the KeyStore and KeyChain APIs. As a hardware-based security module, StrongBox enhances protection for cryptographic keys and credentials managed by these APIs, thereby elevating overall security levels. For devices without StrongBox, the KeyStore and KeyChain APIs still provide significant security improvements through software-based mechanisms like the Trusted Execution Environment (TEE). Given the anticipated migration toward StrongBox-equipped devices, proactive implementation of these APIs is strongly recommended. This approach effectively mitigates security risks associated with hardcoding while enabling robust, future-proof application development.

5.6.3.8 Mechanism and risks of inter-app key sharing using Key Sharing API

Android 16 adds a Key Sharing API for sharing keys created in KeyStore with other apps. These APIs are provided in the KeyStoreManager class. The app granting the key grants access to the key with grantKeyAccess(String alias, int uid) and revokes access from the key with revokeKeyAccess(String alias, int uid). alias is the alias of the created key, and uid is the uid of the app being granted access to the key. The app receiving the key specifies the key with getGrantedKeyFromId(long id) or getGrantedKeyPairFromId(long id). This id is the return value of grantKeyAccess. Therefore, this id must be passed between the granting app and the grantee app. Keys created using the Android KeyStore provider are inherently intended to be usable only within the app that created them. Therefore, the Key Sharing API may weaken the app's security level. Therefore, when using the Key sharing API, you need to pay attention to the following points and consider your use case before taking security measures and accepting risks.

- Check the trustworthiness of allowed apps: Allowed apps are specified by uid. This means that a fixed value cannot be specified, as it is a different value for each user's device. When specifying a package and checking the uid, the legitimacy of the installed app must be verified. When linking with allowed apps, it is necessary to implement a verification method by referring to 4.1.2.10. *Verify the Destination Activity if Linking with Another Company's Application (Required)*. Check the destination activity when linking with specific third-party apps (required).
- Managing authorized apps: There is no API provided to obtain the devices that are authorized to use the key. Therefore, if you need to manage authorized apps, you will need to create and maintain your own list
- Permission expiration: You also need to consider how long you want to continue allowing a key (there is no function to automatically revoke permission). After allowing a key, malware could be introduced during an update of the allowed app, or a vulnerability could be discovered and exploited, potentially allowing access to the key in an unexpected way. Therefore, depending on the use case, you may want to consider incorporating logic to revoke permission after a certain period of time has passed, if necessary.

5.6.3.9 (Column) Key Management with Google Play App Signing

Android's Play App Signing (Google Play App Signing) is an important security feature when publishing apps to the Google Play Store. App signing is the process by which developers digitally sign their app's APK file, which proves that the app has not been tampered with. When a user installs an app, the Android system verifies the signature to determine whether the app is trustworthy.

Google Play App Signing is a service that allows developers to delegate app signing to Google. By using this service, Google takes responsibility for app key management and signing, achieving a higher level of security. Developers digitally sign only the APK they initially upload, and subsequent version management and signing are handled by Google.

The signing process begins with the developer creating a keystore on their machine and then digitally signing the APK file with the private key stored in that keystore. This can be done using Android Studio or command line tools. The signed APK file is then uploaded to the Google Play console, and the app is ready for publication. When using the Google Play Signing service, the developer signs only the first upload; subsequent versions are signed by Google.

By using the Google Play Signing Service, developers can mitigate key management risks, and Google has implemented advanced security measures to ensure keys are secure, protecting apps from unauthorized access and tampering.

Additionally, apps signed with the Google Play Signing Service are more likely to be trusted by users, which can lead to more downloads and positive reviews. Additionally, signed apps are subject to scanning by Google Play Protect for added security.

Android's Play App Signing is an essential feature for ensuring the security and reliability of apps. By utilizing the Google Play Signing Service, developers can reduce the burden of key management and provide users with highly reliable apps. Understanding this mechanism and using it appropriately will lead to the development of secure apps.

There have been reported cases where signing keys have been leaked or misused due to insufficient management. See below for specific examples.

https://gigazine.net/news/20221203-samsung-android-signing-key-leaked/?utm_source=chatgpt.com

https://www.mcafee.com/blogs/other-blogs/mcafee-labs/fakecalls-android-malware-abusing-legitimate-signing-key/

https://www.androidpolice.com/2019/08/29/cryptographic-key-used-to-sign-one-of-facebooks-android-apps-compromised/?utm_source=chatgpt.com

5.6.3.10 Per-App Android Keystore (Key) Limits

In Android 17, a limit on the number of keys in Android Keystore that can be used per app was introduced for the purpose of improving platform stability and security. Specifically, it is a restriction where non-system apps can create a maximum of up to 50,000 keys (entries).

This limit applies only to keys stored in the AndroidKeyStore system provided by Android. Keys generated using KeyGenParameterSpec or similar and saved in the AndroidKeyStore provider are counted, and AES keys, RSA key pairs, EC keys, etc. are all counted as one key per alias unit. On the other hand, file-based keystores in BKS or JKS format saved in the app's internal storage, and keys managed by unique cryptographic libraries, are not included in this limit. It is positioned solely as a resource limit on the AndroidKeyStore area managed by the OS.

Behind this change is the aim to prevent the issue where apps unintentionally continue generating large numbers of keys, putting pressure on device-wide resources. In Android 17, multiple enhancements to memory and resource management have been introduced, and as part of this, a limit has also been set on Keystore usage. Attempting to create keys exceeding the limit results in a KeyStoreException, causing creation to fail.

In practice, for many general apps, the number of keys remains within several to several dozen, so there are hardly any cases where this limit is reached. However, depending on the design, caution is required. For example, in designs where individual keys are generated for each user, or implementations where a new alias is created for each login or process, keys increase in proportion to the number of users or process executions, creating a risk of eventually reaching the limit.

Therefore, in Android 17 or later, the following design is recommended. First, store a small number of master keys shared across the entire app in Keystore, and use temporary keys or data keys generated inside the app for encrypting other data. Then, by encrypting (wrapping) those data keys with the master key, both security and scalability can be achieved. With this method, the number of keys in Keystore is kept almost constant, and there is no concern of reaching the limit.

Although it is unlikely to become an issue in standard implementations, depending on how keys are used, there is a possibility of reaching the limit, so it is important to review the design of alias reuse and key management. In particular, checking whether the implementation generates keys each time is an important point in future Android compatibility.

5.7 Using biometric authentication features

A variety of methods for biological authentication are currently under research and development, with methods using facial information and vocal signatures particularly prominent. Among these methods, methods for using fingerprint authentication to identify individuals have been used since ancient times, and are used today for purposes such as

signatures (by thumbprint) and crime investigation. Applications of fingerprinting have also advanced in several areas of the computer world, and in recent years these methods have begun to enjoy wide recognition as highly convenient techniques (offering advantages such as ease of input) for use in areas such as identifying the owner of a smartphone (primarily for unlocking screens).

Capitalizing on these trends, Android 6.0(API Level 23) incorporates a framework for fingerprint authentication on terminals, which allows apps to make use of fingerprint authentication features(FingerprintManager) to identify individuals. Also, in Android 9.0 (API level 28), a BiometricPrompt API was added for providing comprehensive support for face recognition, iris recognition, and other biometric recognition functions beyond just simply fingerprint authentication. Also, the authentication UI that previously had to be provided separately by the app is no longer needed, and a standard authentication dialog box is automatically used instead. Together with this change, the previous fingerprint authentication function (FingerprintManager) was deprecated. In what follows we discuss some security precautions to keep in mind when using BiometricPrompt authentication.

5.7.1 Sample Code

In biometric authentication functions, there are two major use cases: when a key linked to the user's authentication information is used and when simply performing user authentication only. Based on the application of this biometric authentication, select the sample code based on Fig. 5.7.1.

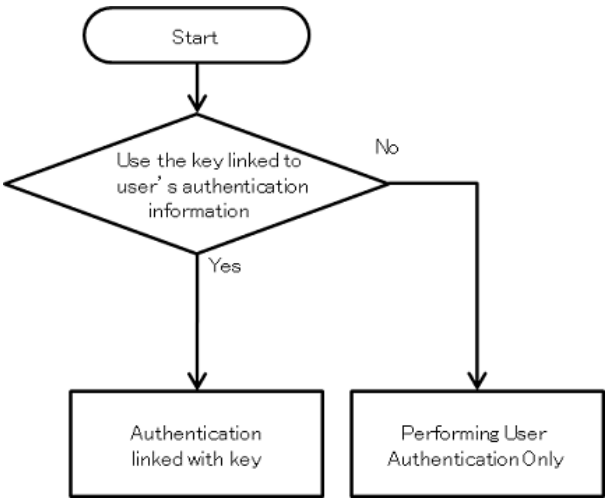


Fig. 5.7.1: Selection flowchart for sample code using biometric authentication

At the time when Android 9.0 (API level 29) was released, no BiometricPrompt support library was available, and so this meant that usage was limited to devices running Android 9.0 only. However, currently, the support library [androidx.biometric](https://developer.android.com/reference/androidx/biometric/package-summary) is available, and this enables the use of BiometricPrompt in a wide range of models from Android 6.0 and higher. The sample code shown below uses BiometricPrompt, which is provided as a support library.

5.7.1.1 Authentication Linked with Key

We present sample code below that allows an application to use Android's biometric authentication feature.

Points:

1. Declare the use of the USE_FINGERPRINT(Android 6.0 - Android 8.1) or USE_BIOMETRIC(Android 9.0 -) permission[38].

[38] In Android 6.0 (API level 23) to Android 8.1 (API level 27) devices, for the BiometricPrompt of the support library androidx.biometric that is used in the sample code, use of the supported USE_FINGERPRINT permission must be declared in order to use the FingerPrintManager function and perform biometric (fingerprint) authentication. By contrast, in devices running Android 9.0 (API level 28) or higher, the BiometricPrompt function of android.hardware.biometrics is used, and use of the USE_BIOMETRIC permission must be declared (In actuality, the use of these permissions has already been declared in the manifest file of the support library package androidx.biometric, and so the manifest file at the app side that uses it can run without any problems even if use was not declared).

2. Obtain an instance from the “AndroidKeyStore” Provider.
3. Notify users that biometric registration will be required to create a key.
4. When creating (registering) keys, use an encryption algorithm that is not vulnerable (meets standards).
5. When creating (registering) keys, enable requests for user (biometric) authentication (do not specify the duration over which authentication is enabled).
6. Design your application on the assumption that the status of biometric registration will change between when keys are created and when keys are used.
7. Restrict encrypted data to items that can be restored (replaced) by methods other than biometric authentication.

MainActivity.java

```java
/*
 * Copyright (C) 2012-2026 Japan Smartphone Security Association
 *
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 *
 *      http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */

package org.jssec.android.biometricprompt.cipher;

import androidx.appcompat.app.AlertDialog;
import androidx.biometric.BiometricPrompt;

import android.app.KeyguardManager;
import android.content.Context;
import android.content.pm.PackageManager;
import android.icu.text.SimpleDateFormat;
import android.os.Build;
import android.os.Bundle;
import android.util.Base64;
import android.view.View;
import android.widget.Button;
import android.widget.TextView;
import androidx.appcompat.app.AppCompatActivity;
import java.util.Date;
import javax.crypto.BadPaddingException;
import javax.crypto.Cipher;
import javax.crypto.IllegalBlockSizeException;

public class MainActivity extends AppCompatActivity {
    private BiometricAuthentication mBiometricAuthentication;
    private static final String SENSITIVE_DATA = "sensitive date";

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
```

(continues on next page)

(continued from previous page)

```
        if (!isBiometricEnabled(this)) {
            // *** POINT 3 *** Notify users that biometric information
            // registration will be required to create a key
            new AlertDialog.Builder(this)
                .setTitle(R.string.app_name)
                .setMessage("No biometric information has been registered. \n" +
                "Click \"Security\" on the Settings menu to register fingerprints.␣
↪\n" +
                "Registering biometric information allows easy authentication.")
                .setPositiveButton("OK", null)
                .show();
            return;
        }

        // Callback which receives the result of biometric authentication
        BiometricPrompt.AuthenticationCallback callback =
                new BiometricPrompt.AuthenticationCallback() {
            @Override
            public void onAuthenticationError(int errorCode,
                                              CharSequence errString) {
                showMessage(errString, R.color.colorError);
            }

            @Override
            public void onAuthenticationSucceeded(
                        BiometricPrompt.AuthenticationResult result) {
                Cipher cipher = result.getCryptoObject().getCipher();
                try {
                    // *** POINT 7 *** Limit encrypted data to items that can be
                    // restored (replaced) by methods other than fingerprint
                    // authentication
                    byte[] encrypted = cipher.doFinal(SENSITIVE_DATA.getBytes());
                    showEncryptedData(encrypted);
                } catch (IllegalBlockSizeException | BadPaddingException e) {
                }

                showMessage(getString(R.string.biometric_auth_succeeded),
                            R.color.colorAuthenticated);
                reset();
            }

            @Override
            public void onAuthenticationFailed() {
                showMessage(getString(R.string.biometric_auth_failed),
                             R.color.colorError);
            }
        };

        mBiometricAuthentication =
                new BiometricAuthentication(this, callback);

        Button button_biometric_auth = findViewById(R.id.button_biometric_auth);
        button_biometric_auth.setOnClickListener(new View.OnClickListener () {
            @Override
            public void onClick(View v) {
```

(continues on next page)

(continued from previous page)

```
                if (mBiometricAuthentication.startAuthentication()) {
                    showEncryptedData(null);
                }
            }
        });
    }

    private Boolean isBiometricEnabled(Context con) {
        return Build.VERSION.SDK_INT >= Build.VERSION_CODES.M &&
            con.getSystemService(KeyguardManager.class).isKeyguardSecure() &&
            con.getPackageManager()
               .hasSystemFeature(PackageManager.FEATURE_FINGERPRINT);
    }

    private void setAuthenticationState(boolean authenticating) {
        Button button = (Button) findViewById(R.id.button_biometric_auth);
        button.setText(authenticating ? R.string.cancel : R.string.authenticate);
    }

    private void showEncryptedData(byte[] encrypted) {
        TextView textView = (TextView) findViewById(R.id.encryptedData);
        if (encrypted != null) {
            textView.setText(Base64.encodeToString(encrypted, 0));
        } else {
            textView.setText("");
        }
    }

    private String getCurrentTimeString() {
        long currentTimeMillis = System.currentTimeMillis();
        Date date = new Date(currentTimeMillis);
        SimpleDateFormat simpleDateFormat = new SimpleDateFormat("HH:mm:ss.SSS");

        return simpleDateFormat.format(date);
    }

    private void showMessage(CharSequence msg, int colorId) {
        TextView textView = (TextView) findViewById(R.id.textView);
        textView.setText(getCurrentTimeString() + " :\n" + msg);
        textView.setTextColor(getResources().getColor(colorId, null));
    }

    private void reset() {
        setAuthenticationState(false);
    }
}
```

BiometricAuthentication.java

```
/*
 * Copyright (C) 2012-2026 Japan Smartphone Security Association
 *
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 *
 *      http://www.apache.org/licenses/LICENSE-2.0
```

(continues on next page)

(continued from previous page)

```
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */

package org.jssec.android.biometricprompt.cipher;

import android.app.KeyguardManager;
import android.content.Context;
import android.content.DialogInterface;
import android.os.CancellationSignal;
import android.os.Handler;
import android.security.keystore.KeyGenParameterSpec;
import android.security.keystore.KeyInfo;
import android.security.keystore.KeyPermanentlyInvalidatedException;
import android.security.keystore.KeyProperties;

import androidx.biometric.BiometricPrompt;
import androidx.fragment.app.FragmentActivity;

import java.io.IOException;
import java.security.InvalidAlgorithmParameterException;
import java.security.InvalidKeyException;
import java.security.KeyStore;
import java.security.KeyStoreException;
import java.security.NoSuchAlgorithmException;
import java.security.NoSuchProviderException;
import java.security.UnrecoverableKeyException;
import java.security.cert.CertificateException;
import java.security.spec.InvalidKeySpecException;
import java.util.concurrent.Executor;

import javax.crypto.Cipher;
import javax.crypto.KeyGenerator;
import javax.crypto.NoSuchPaddingException;
import javax.crypto.SecretKey;
import javax.crypto.SecretKeyFactory;

public class BiometricAuthentication {
    private static final String TAG = "BioAuth";

    private static final String KEY_NAME = "KeyForFingerprintAuthentication";
    private static final String PROVIDER_NAME = "AndroidKeyStore";
    private androidx.biometric.BiometricPrompt mBiometricPrompt;
    private androidx.biometric.BiometricPrompt.PromptInfo mPromptInfo;
    private CancellationSignal mCancellationSignal;
    private KeyStore mKeyStore;
    private KeyGenerator mKeyGenerator;
    private Cipher mCipher;


    public BiometricAuthentication(FragmentActivity context, final androidx.
↪biometric.BiometricPrompt.AuthenticationCallback callback) {
```

(continues on next page)

(continued from previous page)

```
        // Callback which receives the result of biometric authentication
        androidx.biometric.BiometricPrompt.AuthenticationCallback hook =
                new androidx.biometric.BiometricPrompt.AuthenticationCallback() {
            @Override
            public void onAuthenticationError(int errorCode,
                                              CharSequence errString) {
                android.util.Log.e(TAG, "onAuthenticationError");
                if (callback != null) {
                    callback.onAuthenticationError(errorCode, errString);
                }
                reset();
            }

            @Override
            public void onAuthenticationSucceeded(androidx.biometric.
↪BiometricPrompt.AuthenticationResult result) {
                android.util.Log.e(TAG, "onAuthenticationSuccess");
                if (callback != null) {
                    callback.onAuthenticationSucceeded(result);
                }
                reset();
            }

            @Override
            public void onAuthenticationFailed() {
                android.util.Log.e(TAG, "onAuthenticationFailed");
                if (callback != null) {
                    callback.onAuthenticationFailed();
                }
            }
        };

        final Handler mHandler = new Handler(context.getMainLooper());
        final Executor mExecutor = new Executor() {
            @Override
            public void execute(Runnable runnable) {
                mHandler.post(runnable);
            }
        };

        mBiometricPrompt =
            new androidx.biometric.BiometricPrompt(context, mExecutor, hook);
        final androidx.biometric.BiometricPrompt.PromptInfo.Builder builder =
            new androidx.biometric.BiometricPrompt.PromptInfo.Builder()
                .setTitle("Please Authenticate")
                .setNegativeButtonText("Cancel");
        mPromptInfo = builder.build();
        reset();
    }

    public boolean startAuthentication() {
        if (!generateAndStoreKey())
            return false;

        if (!initializeCipherObject())
            return false;
```

(continues on next page)

(continued from previous page)

```
        androidx.biometric.BiometricPrompt.CryptoObject cryptoObject =
            new BiometricPrompt.CryptoObject(mCipher);

        //  Process biometric authentication
        android.util.Log.e(TAG, "Starting authentication");
        mBiometricPrompt.authenticate(mPromptInfo, cryptoObject);
        return true;
    }

    private void reset() {
        try {
            // *** POINT 2 ** Obtain an instance from the
            // “AndroidKeyStore” Provider.
            mKeyStore = KeyStore.getInstance(PROVIDER_NAME);
            mKeyGenerator =
                KeyGenerator.getInstance(KeyProperties.KEY_ALGORITHM_AES,
                                         PROVIDER_NAME);
            mCipher =
                Cipher.getInstance(KeyProperties.KEY_ALGORITHM_AES
                                   + "/" + KeyProperties.BLOCK_MODE_CBC
                                   + "/" + KeyProperties.ENCRYPTION_PADDING_PKCS7);
        } catch (KeyStoreException | NoSuchPaddingException
                 | NoSuchAlgorithmException | NoSuchProviderException e) {
            throw new RuntimeException("failed to get cipher instances", e);
        }
        mCancellationSignal = null;
    }

    private boolean generateAndStoreKey() {
        try {
            mKeyStore.load(null);
            if (mKeyStore.containsAlias(KEY_NAME))
                mKeyStore.deleteEntry(KEY_NAME);
            mKeyGenerator.init(
                // *** POINT 4 *** When creating (registering) keys,
                // use an encryption algorithm that is not vulnerable
                // (meets standards)
                new KeyGenParameterSpec.Builder(KEY_NAME, KeyProperties.PURPOSE_
↪ENCRYPT)
                    .setBlockModes(KeyProperties.BLOCK_MODE_CBC)
                    .setEncryptionPaddings(KeyProperties.ENCRYPTION_PADDING_PKCS7)
                    // *** POINT 5 *** When creating (registering) keys, enable
                    // requests for user (fingerprint) authentication (do not
                    // specify the duration over which authentication is enabled)
                    .setUserAuthenticationRequired(true)
                    .build());
            // Generate a key and store it to Keystore(AndroidKeyStore)
            mKeyGenerator.generateKey();
            return true;
        } catch (IllegalStateException e) {
            return false;
        } catch (NoSuchAlgorithmException | InvalidAlgorithmParameterException
                 | CertificateException | KeyStoreException | IOException e) {
            android.util.Log.e(TAG, "key generation failed: " + e.getMessage());
            throw new RuntimeException("failed to generate a key", e);
```

(continues on next page)

(continued from previous page)

```
        }
    }

    private boolean initializeCipherObject() {
        try {
            mKeyStore.load(null);
            SecretKey key = (SecretKey) mKeyStore.getKey(KEY_NAME, null);
            SecretKeyFactory factory =
                SecretKeyFactory.getInstance(KeyProperties.KEY_ALGORITHM_AES,
                                             PROVIDER_NAME);
            KeyInfo info = (KeyInfo) factory.getKeySpec(key, KeyInfo.class);

            mCipher.init(Cipher.ENCRYPT_MODE, key);
            return true;
        } catch (KeyPermanentlyInvalidatedException e) {
            // *** POINT 6 *** Design your application on the assumption that
            // the status of fingerprint registration will change between
            // when keys are created and when keys are used
            return false;
        } catch (KeyStoreException | CertificateException
                 | UnrecoverableKeyException | IOException
                 | NoSuchAlgorithmException | InvalidKeySpecException
                 | NoSuchProviderException | InvalidKeyException e) {
            android.util.Log.e(TAG, "failed to init Cipher: " + e.getMessage());
            throw new RuntimeException("failed to init Cipher", e);
        }
    }
}
```

AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
          >

  <!--  *** POINT 1 *** Declare the use of the USE_BIOMETRIC permission -->
  <uses-permission android:name="android.permission.USE_BIOMETRIC" />

  <application
      android:allowBackup="true"
      android:icon="@mipmap/ic_launcher"
      android:label="@string/app_name"
      android:roundIcon="@mipmap/ic_launcher_round"
      android:supportsRtl="true"
      android:theme="@style/Theme.AppCompat.Light">
    <activity android:name="org.jssec.android.biometricprompt.cipher.MainActivity"
        android:exported="true">
      <intent-filter>
        <action android:name="android.intent.action.MAIN" />

        <category android:name="android.intent.category.LAUNCHER" />
      </intent-filter>
    </activity>
  </application>

</manifest>
```

5.7.1.2 Performing User Authentication Only

The sample code for using biometric authentication when user authentication only is performed is shown below. In this case, you do not need to pay attention to any particular security points, but the sample code is provided below for reference.

Example using BiometricPrompt

MainActivity.java

```
/*
 * Copyright (C) 2012-2026 Japan Smartphone Security Association
 *
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 *
 *      http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */

package org.jssec.android.biometricprompt.nocipher;

import android.hardware.biometrics.BiometricPrompt;
import android.icu.text.SimpleDateFormat;
import androidx.appcompat.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.TextView;
import org.jssec.android.biometric.authentication.nocipher.R;
import java.util.Date;


public class MainActivity extends AppCompatActivity {
    private BiometricAuthentication mBiometricAuthentication;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        mBiometricAuthentication = new BiometricAuthentication(this);

        Button button_biometric_auth = findViewById(R.id.button_biometric_auth);
        button_biometric_auth.setOnClickListener(new View.OnClickListener () {
            @Override
            public void onClick(View v) {
                if (!mBiometricAuthentication.isAuthenticating()) {
                    authenticateByBiometric();
                }
            }
        });
    }
```

(continues on next page)

(continued from previous page)

```
    private boolean authenticateByBiometric () {

        BiometricPrompt.AuthenticationCallback callback =
                new BiometricPrompt.AuthenticationCallback() {
            @Override
            public void onAuthenticationError(int errorCode,
                                              CharSequence errString) {
                showMessage(errString, R.color.colorError);
            }

            @Override
            public void onAuthenticationHelp(int helpCode,
                                             CharSequence helpString) {
                showMessage(helpString, R.color.colorHelp);
            }

            @Override
            public void onAuthenticationSucceeded(
                    BiometricPrompt.AuthenticationResult result) {
                showMessage(getString(R.string.biometric_auth_succeeded),
                            R.color.colorAuthenticated);
            }

            @Override
            public void onAuthenticationFailed() {
                showMessage(getString(R.string.biometric_auth_failed),
                             R.color.colorError);
            }

        };
        if (mBiometricAuthentication.startAuthentication(callback)) {
            showMessage(getString(R.string.biometric_processing),
                        R.color.colorNormal);
            return true;
        }
        return false;
    }

    private String getCurrentTimeString() {
        long currentTimeMillis = System.currentTimeMillis();
        Date date = new Date(currentTimeMillis);
        SimpleDateFormat simpleDateFormat = new SimpleDateFormat("HH:mm:ss.SSS");

        return simpleDateFormat.format(date);
    }

    private void showMessage(CharSequence msg, int colorId) {
        TextView textView = (TextView) findViewById(R.id.textView);
        textView.setText(getCurrentTimeString() + " :\n" + msg);
        textView.setTextColor(getResources().getColor(colorId, null));
    }
}
```

BiometricAuthentication.java

```
/*
```

(continues on next page)

(continued from previous page)

```
 * Copyright (C) 2012-2026 Japan Smartphone Security Association
 *
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 *
 *      http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */

package org.jssec.android.biometricprompt.nocipher;

import android.content.Context;
import android.content.DialogInterface;
import android.hardware.biometrics.BiometricPrompt;
import android.os.CancellationSignal;

public class BiometricAuthentication {
    private static final String TAG = "BioAuth";

    private BiometricPrompt mBiometricPrompt;
    private CancellationSignal mCancellationSignal;
    private Context mContext;

    // Process "Cancel" button
    private DialogInterface.OnClickListener cancelListener =
        new DialogInterface.OnClickListener () {
        @Override
        public void onClick(DialogInterface dialog, int which) {
            android.util.Log.d(TAG, "cancel");
            if (mCancellationSignal != null) {
                if (!mCancellationSignal.isCanceled())
                    mCancellationSignal.cancel();
            }
        }
    };

    public BiometricAuthentication(Context context) {
        mContext = context;
        BiometricPrompt.Builder builder = new BiometricPrompt.Builder(context);
        // Authentication prompt also provides a button for cacelling
        // Cancel is handled by DialogInterface.OnClickListener
        // given to setNegativeButton as the 3rd argument
        mBiometricPrompt = builder
            .setTitle("Please Authenticate")
            .setNegativeButton("Cancel",context.getMainExecutor() ,cancelListener)
            .build();
        reset();
    }

    public boolean startAuthentication(
```

(continues on next page)

(continued from previous page)

```
            final BiometricPrompt.AuthenticationCallback callback) {

        mCancellationSignal = new CancellationSignal();

        // Callback which accepts the result of biometric authentication
        BiometricPrompt.AuthenticationCallback hook =
            new BiometricPrompt.AuthenticationCallback() {
                @Override
                public void onAuthenticationError(int errorCode,
                                                  CharSequence errString) {
                    android.util.Log.d(TAG, "onAuthenticationError");
                    if (callback != null) {
                        callback.onAuthenticationError(errorCode, errString);
                    }
                    reset();
                }

                @Override
                public void onAuthenticationHelp(int helpCode,
                                                 CharSequence helpString) {
                    android.util.Log.d(TAG, "onAuthenticationHelp");
                    if (callback != null) {
                        callback.onAuthenticationHelp(helpCode, helpString);
                    }
                }

                @Override
                public void onAuthenticationSucceeded(
                        BiometricPrompt.AuthenticationResult result) {
                    android.util.Log.d(TAG, "onAuthenticationSuccess");
                    if (callback != null) {
                        callback.onAuthenticationSucceeded(result);
                    }
                    reset();
                }

                @Override
                public void onAuthenticationFailed() {
                    android.util.Log.d(TAG, "onAuthenticationFailed");
                    if (callback != null) {
                        callback.onAuthenticationFailed();
                    }
                }
            };

        // Perform biomettic authentication
        // BiometricPrompt has a specific API for simple authentication
        // (not linked with key)
        android.util.Log.d(TAG, "Starting authentication");
        mBiometricPrompt.authenticate(mCancellationSignal,
                                      mContext.getMainExecutor(),
                                      hook);

        return true;
    }
```

(continues on next page)

(continued from previous page)

```
    public boolean isAuthenticating() {
        return mCancellationSignal != null && !mCancellationSignal.isCanceled();
    }

    private void reset() {
        mCancellationSignal = null;
    }
}
```

```
AndroidManifest.xml
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
          >

  <!--  *** POINT 1 *** Declare the use of the USE_BIOMETRIC permission -->
  <uses-permission android:name="android.permission.USE_BIOMETRIC" />

  <application
      android:allowBackup="true"
      android:icon="@mipmap/ic_launcher"
      android:label="@string/app_name"
      android:roundIcon="@mipmap/ic_launcher_round"
      android:supportsRtl="true"
      android:theme="@style/AppTheme">
    <activity android:name="org.jssec.android.biometricprompt.nocipher.MainActivity
↪"
        android:exported="true">
      <intent-filter>
        <action android:name="android.intent.action.MAIN" />

        <category android:name="android.intent.category.LAUNCHER" />
      </intent-filter>
    </activity>
  </application>

</manifest>
```

5.7.2 Rule Book

Observe the following rules when using biometric authentication. There are no particular rules when using the fingerprint authentication function for other applications.

1. *When creating (registering) keys, use an encryption algorithm that is not vulnerable (meets standards). (Required)*
2. *Restrict encrypted data to items that can be restored (replaced) by methods other than biometric authentication. (Required)*
3. *Notify users that biometric information registration will be required to create a key. (Recommended)*

5.7.2.1 When creating (registering) keys, use an encryption algorithm that is not vulnerable (meets standards). (Required)

Like the password keys and public keys discussed in Section "5.6. *Using Cryptography*", when using biometric authentication features to create keys it is necessary to use encryption algorithms that are not vulnerable---that is, algorithms that meet certain standards adequate to prevent eavesdropping by third parties. Indeed, safe and non-vulnerable choices must be made not only for encryption algorithms but also for encryption modes and padding.

For more information on selecting algorithms, see Section "5.6.2.2. *Use Strong Algorithms (Specifically, Algorithms that Meet the Relevant Criteria) (Required)*".

5.7.2.2 Restrict encrypted data to items that can be restored (replaced) by methods other than biometric authentication. (Required)

When an app uses biometric authentication features for the encryption of data within the app, the app must be designed in such a way as to allow the data to be recovered (replaced) by methods other than biometric authentication.

In general, the use of biological information entails various problems---including secrecy, the difficulty of making modifications, and erroneous identifications---and it is thus best to avoid relying solely on biological information for authentication.

For example, suppose that data internal to an app is encrypted with a key generated using biometric authentication features, but that the iometric data stored within the terminal is subsequently deleted by the user. Then the key used to encrypt the data is not available for use, nor is it possible to copy the data. If the data cannot be recovered by some means other than biometric-authentication functionality, there is substantial risk that the data will be made useless.

Moreover, the deletion of biometric information is not the only scenario in which keys created using biometric authentication functions can become unusable. In Nexus5X, if biometric authentication features are used to create a key and this key is then newly registered as an addition to the biometric information, keys created earlier have been observed to become unusable.

5.7.2.3 Notify users that biometric information registration will be required to create a key. (Recommended)

In order to create a key using biometric authentication, it is necessary that a user's biometrics be registered on the terminal. When designing apps to guide users to the Settings menu to encourage biometric registration, developers must keep in mind that biometrics represent important personal data, and it is desirable to explain to users why it is necessary or convenient for the app to use biometric information.

Notify users the fingerprint registration will be required.

```
if (!mFingerprintAuthentication.isFingerprintAuthAvailable()) {
    // *** Point *** Notify users that biometric registration will be
    // required to create a key.
    new AlertDialog.Builder(this)
        .setTitle(R.string.app_name)
        .setMessage("No biometric information has been registered.\n" +
            "Click \"Security\" on the Settings menu to register biometrics.\n" +
            "Registering biometrics allows easy authentication.")
        .setPositiveButton("OK", null)
        .show();
    return false;
}
```

5.7.3 Advanced Topics

5.7.3.1 Preconditions for the use of biometric authentication features by Android apps

The following two conditions must be satisfied in order for an app to use biometric authentication.

- User biometrics must be registered within the terminal.
- An (application-specific) key must be associated with registered biometrics.

Registering user biometrics

User biometric information can only be registered via the "Security" option in the Settings menu; ordinary applications may not perform the biometric registration procedure. For this reason, if no biometrics have been registered when an app attempts to use biometric authentication features, the app must guide the user to the Settings menu and encourage the user to register biometrics. At this time, it is desirable for the app to offer the user some explanation of why it is necessary and convenient to use biometric information.

In addition, as a necessary precondition for biometric registration to be possible, the terminal must be configured with an alternative screen-locking mechanism. If the screen lock is disabled in a state in which biometric have been registered in the terminal, the registered biometric information will be deleted.

Creating and registering keys

To associate a key with biometrics registered in a terminal, use a KeyStore instance provided by an "AndroidKeyStore" Provider to create and register a new key or to register an existing key.

To create a key associated with biometric information, configure the parameter settings when creating a KeyGenerator to enable requests for user authentication.

Creating and registering a key associated with biometric information.

```
try {
    // Obtain an instance from the "AndroidKeyStore" Provider.
    KeyGenerator keyGenerator =
        KeyGenerator.getInstance(KeyProperties.KEY_ALGORITHM_AES,
                                 "AndroidKeyStore");
    keyGenerator.init(
        new KeyGenParameterSpec.Builder(KEY_NAME,
                                        KeyProperties.PURPOSE_ENCRYPT)
        .setBlockModes(KeyProperties.BLOCK_MODE_CBC)
        .setEncryptionPaddings(KeyProperties.ENCRYPTION_PADDING_PKCS7)
        // Enable requests for user (biometric) authentication.
        .setUserAuthenticationRequired(true)
        .build());
    keyGenerator.generateKey();
} catch (IllegalStateException e) {
    // no biometrics have been registered in this terminal.
    throw new RuntimeException("No biometric registered", e);
} catch (NoSuchAlgorithmException | InvalidAlgorithmParameterException
            | CertificateException | KeyStoreException | IOException e) {
    // failed to generate a key.
    throw new RuntimeException("Failed to generate a key", e);
}
```

To associate biometric information with an existing key, register the key with a KeyStore entry to which has been added a setting to enable user authentication requests.

Associating biometric information with an existing key.

```
SecretKey key = existingKey;  // existing key

KeyStore keyStore = KeyStore.getInstance("AndroidKeyStore");
keyStore.load(null);
keyStore.setEntry(
    "alias_for_the_key",
    new KeyStore.SecretKeyEntry(key),
    new KeyProtection.Builder(KeyProperties.PURPOSE_ENCRYPT)
        // Enable requests for user (biometric) authentication.
        .setUserAuthenticationRequired(true)
        .build());
```

5.7.3.2 Changes to Biometric Authentication on Android 11

The following 3 points have been changed for biometric authentication on Android 11.

- Introduction of the BiometricManager.Authenticators interface
- Enhancement of data access in BiometricPrompt
- Method for end of support

BiometricManager.Authenticators interface

Authentication types supported in the BiometricManager class and BiometricPrompt class are defined on the BiometricManager.Authenticators interface as follows.

Table 5.7.1: Authenticators Interface

Authentication Type	Description
BIOMETRIC_STRONG	Authentication that uses hardware elements that make the strength level Strong
BIOMETRIC_WEAK	Authentication that uses hardware elements that make the strength level Weak
DEVICE_CREDENTIAL	Authentication that uses authentication information (user PIN, pattern, password) of screen lock

Pass the above authentication type as an argument to the setAllowedAuthenticators() method and define which authentication type the app accepts. One or more authentication types can be passed. For example, when defining to accept strength level Strong hardware elements and screen lock authentication information, pass BIOMETRIC_STRONG | DEVICE_CREDENTIAL as an argument.

To confirm whether or not authentication elements that apps require can be used, do so through the canAuthenticate() method. At this time, if the PIN, pattern, and password have not been created by the user, call the ACTION_BIOMETRIC_ENROLL intent action. This intent asks the user to register authentication information of the authentication system that the app accepts.

After user authentication, executing the getAuthenticationType() method enables confirmation of the authentication type (device authentication information or biometric authentication information) used by the user.

Enhancement of data access in BiometricPrompt

You can provide support for auth-per-use keys within your instance of BiometricPrompt. Such a key requires the user to present either a biometric credential or a device credential each time your app needs to access data that's guarded by that key. Auth-per-use keys can be useful for high-value transactions, such as making a large payment or updating a person's health records.

To associate a BiometricPrompt object with an auth-per-use key, add code similar to the following.

```
KeyGenParameterSpec authPerOpKeyGenParameterSpec =
        new KeyGenParameterSpec.Builder(KEY_NAME, KeyProperties.PURPOSE_ENCRYPT)
    // Accept either a biometric credential or a device credential.
    .setUserAuthenticationParameters(0, KeyProperties.AUTH_BIOMETRIC_STRONG |␣
↪KeyProperties.AUTH_DEVICE_CREDENTIAL)
    .build();
```

Deprecated methods

Android 11 deprecates the following methods:

- The setDeviceCredentialAllowed() method.
- The setUserAuthenticationValidityDurationSeconds() method.
- The overloaded version of canAuthenticate() that takes no arguments.

6 Difficult Problems

In Android, there are some problems that it is difficult to assure a security by application implementation due to a specification of Android OS or a function which Android OS provides. By being abused by the malicious third party or used by users carelessly, these functions are always holding risks that may lead to security problems like information leakage. In this chapter, by indicating risk mitigation plans that developers can take against these functions, some topics that needs calling attentions, are picked up as articles.

6.1 Risk of Information Leakage from Clipboard

Copy & paste are the functions which users often use in a casual manner. For example, not a few users use these functions to store curious information or important information to remember in a mail or a web page into a notepad, or to copy and to paste a password from a notepad in which passwords are stored in order not to forget in advance. These are very casual actions at a glance, but actually there's a hidden risk that user handling information may be stolen.

The risk is related to mechanism of copy & paste in Android system. The information which was copied by user or application, is once stored in the buffer called Clipboard. The information stored in Clipboard is distributed to other applications when it is pasted by a user or an application. So there is a risk which leads to information leakage in this Clipboard function. It is because the entity of Clipboard is single in a system and any application can obtain the information stored in Clipboard at any time by using ClipboardManager. It means that all the information which user copied/cut, is leaked out to the malicious application.

Hence, application developers need to take measures to minimize the possibility of information leakage, considering the Android OS specifications.

6.1.1 Sample Code

Roughly speaking, there are two outlooks of counter-measures to mitigate the risk of information leakage form Clipboard.

1. Counter-measure when copying from other applications to your application.
2. Counter-measure when copying from your application to other applications.

Firstly, let us discuss the countermeasure 1 above. Supposing that a user copies character strings from other applications like note pad, Web browser or mailer application, and then paste it to EditText in your application. As it turns out, there's no basic counter-measure to prevent from sensitive information leakage due to copy & paste, in this scenario. Since there's no function in Android to control copy operations by the third party application.

So, regarding the countermeasure 1, there's no method other than explaining users the risk of copying & pasting sensitive information, and just continuing to enlighten users to decrease the actions themselves continuously.

Next discussion is the countermeasure 2 above, supposing that the scenario that a user copies sensitive information displayed in your application. In this case, the sound counter-measure for leakage is to prohibit copying/cutting operations from View (TextView, EditText etc.). If there are no copy/cut functions in View where the sensitive information (like personal information) is input/output, information leakage will never happen from your application via Clipboard.

There are several methods to prohibit copying/cutting. This section herein describes the easy and effective methods: One method is to disable long press View and another method is to delete copy/cut items from menu when selecting character string.

Necessary of counter-measure can be determined as per the flow of Fig. 6.1.1. Fig. 6.1.1, "Input type is fixed to Password attribute" means, the input type is necessarily either of the followings three when application is running. In this case, no counter-measures are required since copy/cut are prohibited as default.

- InputType.TYPE_CLASS_TEXT | InputType.TYPE_TEXT_VARIATION_PASSWORD
- InputType.TYPE_CLASS_TEXT | InputType.TYPE_TEXT_VARIATION_WEB_PASSWORD
- InputType.TYPE_CLASS_NUMBER | InputType.TYPE_NUMBER_VARIATION_PASSWORD

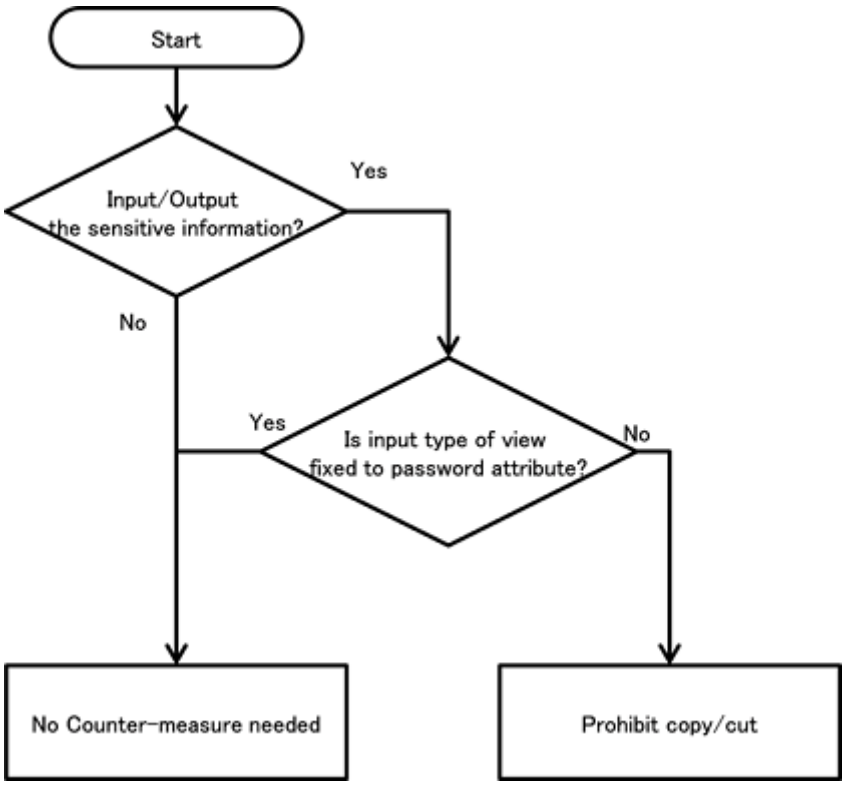


Fig. 6.1.1: Decision flow of counter-measure is required or not

The following subsections detail each countermeasure with sample codes.

6.1.1.1 Delete copy/cut from the menu when character string selection

By TextView.setCustomSelectionActionMODECallback() method, menu when character string selection, can be customized. By using this, if copy/cut item can be deleted from menu when character string selection, user cannot copy/cut character strings any more.

Sample code to delete copy/cut item from menu of character string selection in EditText, is shown as per below.

Points:

1. Delete android.R.id.copy from the menu of character string selection.

2. Delete android.R.id.cut from the menu of character string selection.

UncopyableActivity.java

```
/*
 * Copyright (C) 2012-2026 Japan Smartphone Security Association
 *
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 *
 *      http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */

package org.jssec.android.clipboard.leakage;

import android.app.Activity;
import android.os.Bundle;
import androidx.core.app.NavUtils;
import android.view.ActionMode;
import android.view.Menu;
import android.view.MenuItem;
import android.widget.EditText;

public class UncopyableActivity extends Activity {
    private EditText copyableEdit;
    private EditText uncopyableEdit;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.uncopyable);

        copyableEdit = (EditText) findViewById(R.id.copyable_edit);
        uncopyableEdit = (EditText) findViewById(R.id.uncopyable_edit);
        // By setCustomSelectionActionMODECallback method,
        // Possible to customize menu of character string selection.
        uncopyableEdit.setCustomSelectionActionModeCallback(actionModeCallback);
    }

    private ActionMode.Callback actionModeCallback = new ActionMode.Callback() {
            public boolean onPrepareActionMode(ActionMode mode, Menu menu) {
                return false;
            }

            public void onDestroyActionMode(ActionMode mode) {
            }

            public boolean onCreateActionMode(ActionMode mode, Menu menu) {
                // *** POINT 1 *** Delete android.R.id.copy from the menu of
                // character string selection.
                MenuItem itemCopy = menu.findItem(android.R.id.copy);
                if (itemCopy != null) {
```

(continues on next page)

(continued from previous page)

```
                    menu.removeItem(android.R.id.copy);
                }
                // *** POINT 2 *** Delete android.R.id.cut from the menu of
                // character string selection.
                MenuItem itemCut = menu.findItem(android.R.id.cut);
                if (itemCut != null) {
                    menu.removeItem(android.R.id.cut);
                }
                return true;
            }

            public boolean onActionItemClicked(ActionMode mode, MenuItem item) {
                return false;
            }
        };

    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        getMenuInflater().inflate(R.menu.uncopyable, menu);
        return true;
    }


    @Override
    public boolean onOptionsItemSelected(MenuItem item) {
        switch (item.getItemId()) {
        case android.R.id.home:
            NavUtils.navigateUpFromSameTask(this);
            return true;
        }
        return super.onOptionsItemSelected(item);
    }

}
```

6.1.1.2 Disable Long Click View

Prohibiting copying/cutting can also be realized by disabling Long Click View. Disabling Long Click View can be specified in layout xml file.

Point:

1. Set false to android:longClickable in View to prohibit copy/cut.

```
unlongclickable.xml
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical">

    <TextView
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="@string/unlongclickable_description" />
```

(continues on next page)

(continued from previous page)

```
    <!-- EditText to prohibit copy/cut EditText -->
    <!-- *** POINT 1 *** Set false to android:longClickable in View to prohibit␣
↪copy/cut. -->
    <EditText
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:longClickable="false"
        android:hint="@string/unlongclickable_hint" />
</LinearLayout>
```

6.1.2 Rule Book

Follow the rule below when copying sensitive information from your application to other applications.

1. *Disabling Copy/Cut Character Strings that Are Displayed in View (Required)*

6.1.2.1 Disabling Copy/Cut Character Strings that Are Displayed in View (Required)

If there's a View which displays sensitive information in an application and besides the information is allowed to be copied/cut like EditText in the View, the information may be leaked via Clipboard. Therefore, copy/cut must be disabled in View where sensitive information is displayed.

There are two methods to disable copy/cut. One method is to delete items of copy/cut from menu of character string selection, and another method is to disable Long Click View.

Please refer to "6.1.3.1. *Points to Note When Applying Rules*".

6.1.3 Advanced Topics

6.1.3.1 Points to Note When Applying Rules

Since TextView does not allow text selection by default, countermeasures are usually not required, but depending on the app specifications, copying may be made possible. By using the TextView.setTextIsSelectable() method, whether text can be selected and copied can be configured dynamically. When making TextView copyable, carefully consider whether there is a possibility that sensitive information will be displayed in that TextView, and if there is such a possibility, it should not be made copyable.

Also, as described in the decision flow of "6.1.1. *Sample Code*", for EditText with an input type intended for entering passwords (such as InputType.TYPE_CLASS_TEXT|InputType.TYPE_TEXT_VARIATION_PASSWORD), copying text is prohibited by default, so countermeasures are usually not required. However, if the app uniquely implements features such as "displaying passwords in plain text" and dynamically changes the input type, copying and cutting may become enabled when displaying in plain text, so similar countermeasures are necessary.

Note that when applying the rules, it will also be necessary to consider usability. For example, in the case of a View where users can freely enter text, if copying and cutting are disabled just because the possibility of sensitive information being entered is "not zero", user convenience will deteriorate. Of course, for Views that input/output highly important information or Views where sensitive information is entered on its own, the rules should be applied unconditionally, but when handling other Views, it is good to consider countermeasures while keeping the following points in mind.

- Is it possible to prepare dedicated components for entering or displaying sensitive information?
- If the destination (paste destination) app is known, is it possible to transmit information by another method?
- Is it possible to alert users regarding input and output in the app?
- Is that View really necessary?

Although the lack of security considerations in the specifications of Clipboard and ClipboardManager in Android OS is the root cause that creates the possibility of information leakage, app developers need to handle such Clipboard specifications from various perspectives, such as user protection, usability, and features provided, to create high-quality apps.

6.1.3.2 Usage of Information Stored in Clipboard

The following content is applicable to devices that are Android 9 or earlier. Starting from Android 10, to access the Clipboard, the app must be the default IME or have focus since Clipboard acquisition is limited due to user privacy protections. Verify operation of the sample program on Android 9 devices and earlier[1].

As described in "6.1. *Risk of Information Leakage from Clipboard*", ClipboardManager can be used to access information stored in the clipboard from an app. Also, because ClipboardManager does not require setting of special permissions to use, apps can use ClipboardManager without the user's knowledge.

The information stored in the clipboard (called ClipData) can be obtained by the ClipboardManager.getPrimaryClip() method. ClipData can be obtained without missing any timing because, if an OnPrimaryClipChangedListener is implemented and registered to the ClipboardManager using the ClipboardManager.addPrimaryClipChangedListener() method, the listener will be called each time a copy/cut occurs by user operation or other actions. Here, the listener is called regardless of which app performed the copy/cut.

The following is the source code for a service that obtains ClipData each time a copy/cut occurs in the device and displays it by Toast. We hope you realize that a simple code like the following will make the information stored in the clipboard leak out. When implementing apps, care should be taken to ensure that sensitive information is not obtained, at least by using the following code.

ClipboardListeningService.java

```
/*
 * Copyright (C) 2012-2026 Japan Smartphone Security Association
 *
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 *
 *      http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */

package org.jssec.android.clipboard;

import android.app.Service;
import android.content.ClipData;
import android.content.ClipboardManager;
import android.content.ClipboardManager.OnPrimaryClipChangedListener;
import android.content.Context;
import android.content.Intent;
import android.os.IBinder;
import android.util.Log;
import android.widget.Toast;

public class ClipboardListeningService extends Service {
    private static final String TAG = "ClipboardListeningService";
```

(continues on next page)

[1] https://developer.android.com/about/versions/10/privacy/changes

(continued from previous page)

```
    private ClipboardManager mClipboardManager;

    @Override
    public IBinder onBind(Intent arg0) {
        return null;
    }

    @Override
    public void onCreate() {
        super.onCreate();
        mClipboardManager =
            (ClipboardManager) getSystemService(Context.CLIPBOARD_SERVICE);
        if (mClipboardManager != null) {
            mClipboardManager.addPrimaryClipChangedListener(clipListener);
        } else {
            Log.e(TAG, "Failed to get ClipboardService . Service is closed.");
            this.stopSelf();
        }
    }

    @Override
    public void onDestroy() {
        super.onDestroy();
        if (mClipboardManager != null) {
            mClipboardManager.removePrimaryClipChangedListener(clipListener);
        }
    }

    private OnPrimaryClipChangedListener clipListener =
            new OnPrimaryClipChangedListener() {
        public void onPrimaryClipChanged() {
            if (mClipboardManager != null && mClipboardManager.hasPrimaryClip()) {
                ClipData data = mClipboardManager.getPrimaryClip();
                ClipData.Item item = data.getItemAt(0);
                Toast
                    .makeText(
                        getApplicationContext(),
                        "Character stirng that is copied or cut:\n"
                        + item.coerceToText(getApplicationContext()),
                        Toast.LENGTH_SHORT)
                    .show();
            }
        }
    };
}
```

The following shows an example of source code for an activity using the above ClipboardListeningService.

ClipboardListeningActivity.java

```
/*
 * Copyright (C) 2012-2026 Japan Smartphone Security Association
 *
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 *
```

(continues on next page)

(continued from previous page)

```
 *      http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */

package org.jssec.android.clipboard;

import android.app.Activity;
import android.content.ComponentName;
import android.content.Intent;
import android.os.Bundle;
import android.util.Log;
import android.view.View;

public class ClipboardListeningActivity extends Activity {
    private static final String TAG = "ClipboardListeningActivity";

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_clipboard_listening);
    }

    public void onClickStartService(View view) {
        if (view.getId() != R.id.start_service_button) {
            Log.w(TAG, "View ID is incorrect.");
        } else {
            ComponentName cn =
                startService(new Intent(ClipboardListeningActivity.this,
                             ClipboardListeningService.class));
            if (cn == null) {
                Log.e(TAG, "Failed to launch the service.");
            }
        }
    }

    public void onClickStopService(View view) {
        if (view.getId() != R.id.stop_service_button) {
            Log.w(TAG, "View ID is incorrect.");
        } else {
            stopService(new Intent(ClipboardListeningActivity.this,
                                   ClipboardListeningService.class));
        }
    }
}
```

So far, we have explained how to obtain information stored in the clipboard, but it is also possible to store new information in the clipboard using the ClipboardManager.setPrimaryClip() method.

Note, however, that setPrimaryClip() overwrites the information stored in the clipboard, and so there is a possibility that information stored by the user previously using the copy/cut operation will be lost. If these methods are used to provide a unique copy/cut feature, they should be designed and implemented so that the information stored in the clipboard is not altered in a way that the user does not intend, for example, by displaying a dialog warning that the

information will be altered, if necessary.

Also, if you want to run an app on Android 13 that copies sensitive content such as passwords or credit card information to the clipboard using setPrimaryClip(), you must set the flag in ClipDescription before calling setPrimaryClip(). This is a specification introduced in Android 13 to protect user privacy, and setting this flag prevents the display of sensitive content in content previews. The differences when this flag is set or when it is not set are shown below. This shows an app that uses setPrimaryClip() to copy a password string when the button is pressed.

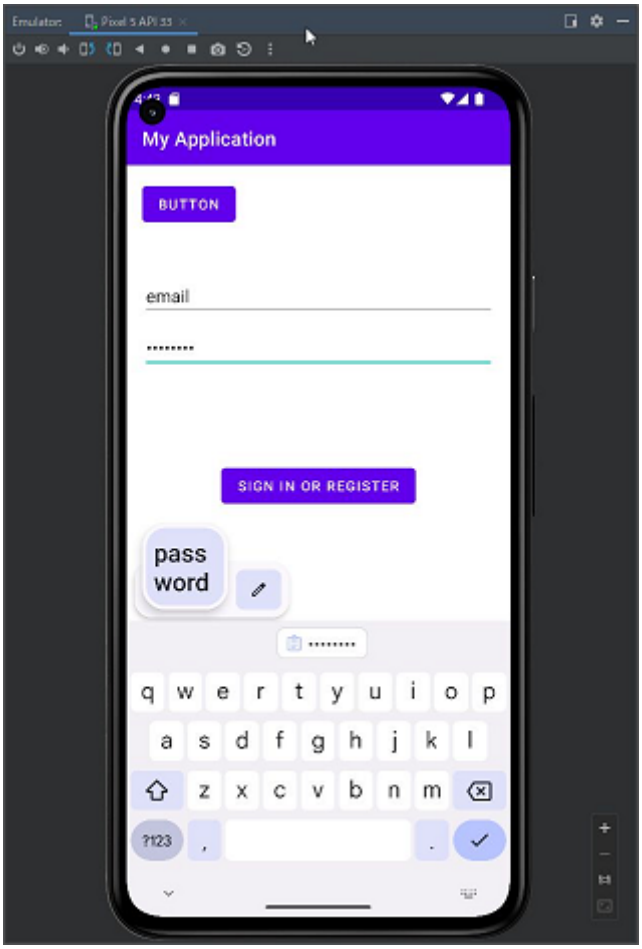


Fig. 6.1.2: Without ClipDescription Flag Setting

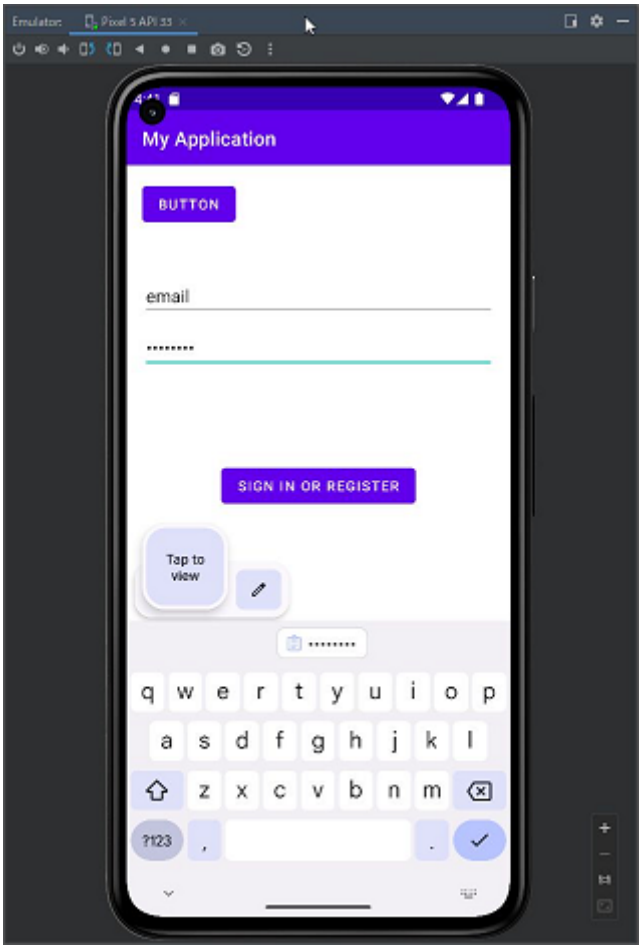


Fig. 6.1.3: With ClipDescription Flag Setting

The method for setting the ClipDescription flag depends on whether the SDK version is API level 33 or higher. The sample code is as follows. When actually setting the ClipDescription flag, match it properly to the SDK version being used from the comments in the code.

```
ClipData.Item item = new ClipData.Item(passwordEditText.getText().toString());
String[] mimeType = new String[1];
mimeType[0] = ClipDescription.MIMETYPE_TEXT_URILIST;
ClipData cd = new ClipData(new ClipDescription("text_data", mimeType), item);
ClipboardManager cm = (ClipboardManager) getSystemService(CLIPBOARD_SERVICE);
PersistableBundle extras = new PersistableBundle();

extras.putBoolean(ClipDescription.EXTRA_IS_SENSITIVE, true); // API level 33 or␣
↪higher
// extras.putBoolean("android.content.extra.IS_SENSITIVE", true); // lower API␣
↪level
cd.getDescription().setExtras(extras);

cm.setPrimaryClip(cd);
```

6.1.3.3 Clipboard Access Notification

From Android 12, a Toast message that notifies users that Clipboard has been accessed is displayed when data stored in Clipboard was acquired on getPrimaryClip() by another application. The notification message is as follows.

```
MYAPP pasted from your clipboard
```

If getPrimaryClipDescription() is used, data is not copied, and the Toast message is not displayed. Information that can be acquired with getPrimaryClipDescription() are truth values that determine whether or not styles have been set for texts as well as information of the stored data itself such as classifications of text (e.g. URL).

Revision history

2014-04-01
Initial English Edition

2014-07-01

Added new articles below

- 5.5. *Handling privacy data*
- 5.6. *Using Cryptography*

2015-06-01

We have reviewed the entire document in accordance with the following policy

- Change of development environment (Eclipse -> Android Studio)
- Responding to Android latest version Lollipop
- Change of API Level (8 or later -> 15 or later)

2016-02-01

Added new articles below

- 4.10. *Using Notifications*
- 5.7. *Using biometric authentication features*

Revised article below

- 5.2. *Permission and Protection Level*

2016-09-01

Revised articles below

- 2.5. *Importing Sample Code into Android Studio*
- 5.4. *Communicating via HTTPS*
- 5.6. *Using Cryptography*

2017-02-01

Added new articles below

- 5.4.3.7. *Network Security Configuration*

Revised articles below

- 4.1. *Creating/Using Activities*

- 4.2. *Receiving/Sending Broadcasts*
- 4.4. *Creating/Using Services*
- 4.5. *Using SQLite*
- 4.6. *Handling Files*

Deleted the section below

- 4.8.3.4 BuildConfig.DEBUG Should Be Used in ADT 21 or Later

We have reviewed the entire document in accordance with the following policy

- All discussions in the main text concerning Android 4.0.3 (API Level 15) and earlier versions have been deleted or moved to footnotes.

2018-02-01

Added new articles below

- 4.1.3.6. *The Autofill framework*
- 5.3.3.2. *Cases in which Authenticator accounts with non-matching signatures may be read in Android 8.0 (API Level 26) or later*
- 5.4.3.8. *(Column): Transitioning to TLS1.2/TLS1.3 for secure connections*
- 5.5.3.3. *Version-dependent differences in handling of Android IDs*

Revised articles below

- 4.2. *Receiving/Sending Broadcasts*
- 5.2. *Permission and Protection Level*
- 5.3. *Add In-house Accounts to Account Manager*
- 5.4. *Communicating via HTTPS*
- 5.5. *Handling privacy data*

2018-09-01

Added new articles below

- 4.9.3.4. *Safe Browsing in WebView*
- 4.11. *Using Shared Memory*

Revised articles below

- 2.5. *Importing Sample Code into Android Studio*
- 4.1.3.6. *The Autofill framework*
- 4.5.3.6. *[Reference] Encrypt SQLite Database (SQLCipher for Android)*
- 5.2.1.2. *How to Communicate Between In-house Applications with In-house-defined Signature Permission*
- 5.4.1.2. *Communicating via HTTPS*
- 5.4.3.7. *Network Security Configuration*
- 5.7. *Using biometric authentication features*
- 5.4.3.2. *Install Root Certificate of Private Certificate Authority to Android OS's Certification Store*
- 5.4.3.8. *(Column): Transitioning to TLS1.2/TLS1.3 for secure connections*

2019-09-01

Added new articles below

- 4.6.3.5. *External storage access for Android 10 and later*

- 5.5.3.4. *Restriction on obtaining non-resettable device identifiers on Android 10*

Revised articles below

- 4.1.3.1. *Combination of Exported Attribute and Intent Filter Setting (For Activity)*
- 4.4.1.2. *Creating/Using Public Services*
- 4.4.3.1. *Combination of Exported Attribute and Intent-filter Setting (In the Case of Service)*
- 4.6.1.4. *Using Eternal Memory (Read Write Public) Files*
- 5.2.3.6. *Modifications to the Permission model specifications in Android versions 6.0 and later*
- 5.4.1.1. *Communicating via HTTP*
- 5.4.1.2. *Communicating via HTTPS*
- 5.4.3.8. *(Column): Transitioning to TLS1.2/TLS1.3 for secure connections*
- 5.5.1.2. *Broad consent is granted: Applications that incorporate application privacy policy*
- 5.6.2.2. *Use Strong Algorithms (Specifically, Algorithms that Meet the Relevant Criteria) (Required)*
- 5.6.3.2. *Generation of random numbers*
- 5.7. *Using biometric authentication features*

2019-12-01

Revised articles below

- 4.1.3.1. *Combination of Exported Attribute and Intent Filter Setting (For Activity)*
- 4.4.1.2. *Creating/Using Public Services*
- 4.4.3.1. *Combination of Exported Attribute and Intent-filter Setting (In the Case of Service)*

2020-11-01

Added new articles below

- 5.2.3.7. *Function That Automatically Resets Unused App Permissions in Android 11.0 and Later*
- 5.5.3.5. *Data Access Auditing*
- 5.5.3.6. *Location Access*
- 5.6.3.5. *Conscrypt Module*
- 5.7.3.2. *Changes to Biometric Authentication on Android 11*

Revised articles below

- 3.1.1.2. *Function Assets of an Android Smartphone*
- 5.2.3.6. *Modifications to the Permission model specifications in Android versions 6.0 and later*

2021-10-19

Added new articles below

- 4.2.3.7. *ACTION_CLOSE_SYSTEM_DIALOGS*
- 4.10.3.2. *Touch to Be Passed Through Specific Window*
- 4.7.3.2. *PendingIntent Mutability*
- 5.5.3.8. *Auto-hibernation Function for Unused Applications on Android 12*
- 5.5.3.9. *API Return Value Change Following Specification Changes to the Package Access*
- 5.5.3.10. *Microphones and Cameras For Android 12*
- 5.5.3.11. *SameSite Cookie on WebView*
- 6.1.3.3. *Clipboard Access Notification*

Revised articles below

- 4.1.3.1. *Combination of Exported Attribute and Intent Filter Setting (For Activity)*
- 4.9.2.3. *Verify That the URL Received from Others Such as Through Intent Is the Expected URL (Required)*
- 5.5.3.5. *Data Access Auditing*
- 5.5.3.6. *Location Access*
- 5.6.3.5. *Conscrypt Module*
- 6.1.3.2. *Usage of Information Stored in Clipboard*

2022-01-17

Revised articles below

- 2.2. *Sample Code, Rule Book, Advanced Topics*

2022-08-29

Added new articles below

- 4.7.3.3. *Strict Enforcement of Intent Filter*
- 4.2.3.8. *Enhanced Safety of Dynamic Broadcast Receiver*
- 4.10.3.3. *Runtime Permissions for Notifications*
- 5.2.3.8. *Revoking Runtime Permissions*
- 4.11.4.2. *Disabling sharedUserId in Newly Installed Apps*

Revised articles below

- 2.4. *Literature on Android Secure Coding*
- 2.5. *Importing Sample Code into Android Studio*
- 5.5.3.6. *Location Access*
- 6.1.3.2. *Usage of Information Stored in Clipboard*

2024-2-29

Added new articles below

- 4.4.3.3. *Requirement for Specifying of Service Types*
- 4.6.3.7. *Enhanced Safety of DCL (Dynamic Code Loading)*
- 4.6.3.9. *Measures for Preventing Path Traversal by Zip Files*
- 4.10.3.4. *Change in Operation for Notifications Indicating Progress*
- 4.6.3.8. *Package Name of Media Owner*

Revised articles below

- 4.1.3.3. *About Root Activity*
- 4.2.3.8. *Enhanced Safety of Dynamic Broadcast Receiver*

2025-1-29

Added new articles below

- 4.7.3.4. *Expanded IntentFilter capabilities*
- 4.7.3.6. *Safer Intents*
- 4.7.3.5. *Changes to PendingIntent and Package Stopped State*
- 4.3.3.1. *Content URI Permission Management*

- 4.4.3.4. *Foreground Service Changes*
- 4.6.3.5. *External storage access for Android 10 and later*
- 4.6.3.6. *Shared storage access for Android 10 and later*
- 4.6.3.10. *Query the most recent user's choice for accessing selected photos*
- 5.5.3.14. *Private Space*
- 5.1.3.5. *Integrate Credential Manager with Autofill*
- 5.1.3.4. *Screen Recording Detection*
- 5.5.3.12. *Partial Screen Sharing*
- 5.5.3.13. *Changes in Global State Management in DND mode*
- 5.6.3.6. *Countermeasures against backup data leaks*
- 5.6.3.7. *Hard-coded Cryptographic Secrets*

Deleted the section below

- 4.6.3.6. About specifications related to access to external storage in Android 10 (API Level 29)
- 4.6.3.7. Application of Scoped Storage in Android 11 (API Level 30)
- 4.6.3.8. Media Collection Permissions in Android 13 (API Level 33)
- 4.6.3.9. Partial Access to Images and Videos in Android 14 (API Level 34)

2025-8-27

Added new articles below

- 4.2.3.9. *Changes to the Priority Specification for Ordered Broadcasts*
- 4.6.3.11. *MediaStore version lockdown*
- 4.6.3.12. *App-owned photos*
- 4.7.3.6. *Safer Intents*
- 4.7.3.7. *Countermeasures against Intent redirect attacks*
- 5.5.3.7. *Local Network Access Permission*
- 5.6.3.8. *Mechanism and risks of inter-app key sharing using Key Sharing API*
- 5.6.3.9. *(Column) Key Management with Google Play App Signing*

Revised articles below

- 4.6.3.4. *Internal storage access for Android 10 and later*
- 4.6.3.5. *External storage access for Android 10 and later*
- 4.6.3.6. *Shared storage access for Android 10 and later*
- 4.7.3.1. *Component Export Control and Intent Sending Restrictions*
- 4.7.3.2. *PendingIntent Mutability*
- 4.7.3.3. *Strict Enforcement of Intent Filter*
- 4.7.3.4. *Expanded IntentFilter capabilities*
- 4.7.3.5. *Changes to PendingIntent and Package Stopped State*
- 5.5.3.14. *Private Space*
- 5.5.3.8. *Auto-hibernation Function for Unused Applications on Android 12*
- 5.5.3.9. *API Return Value Change Following Specification Changes to the Package Access*
- 4.11.4.2. *Disabling sharedUserId in Newly Installed Apps*

- 2.2. *Sample Code, Rule Book, Advanced Topics*
- 5.1.3.5. *Integrate Credential Manager with Autofill*
- 4.6.3.8. *Package Name of Media Owner*
- 5.3.3.2. *Cases in which Authenticator accounts with non-matching signatures may be read in Android 8.0 (API Level 26) or later*

Deleted the section below

- 4.1.3.3. Reading Intents Sent to an Activity
- 4.6.3.4. Specification Change regarding External Storage Access in Android 4.4 (API Level 19) and later
- 4.6.3.5. Revised specifications in Android 7.0 (API Level 24) for accessing specific directories on external storage media
- 4.9.3.1. Vulnerability caused by addJavascriptInterface() at Android versions 4.1 or earlier
- 5.3.3.2. Exception Occurs When Signature Keys of User Application and Authenticator Application Are Different, in Android 4.0.x
- 5.6.3.3. Measures to Protect against Vulnerabilities in Random-Number Generators

2026-9-16

Added new articles below

- 4.1.3.7. *Strengthening Restrictions on Background Activity Launch (BAL)*
- 4.2.3.10. *Autonomous Re-pairing for Bluetooth Bond Loss*
- 4.3.3.2. *Tightening of Contacts Provider (CP2)*
- 4.4.3.5. *Strengthening Background Audio Restrictions*
- 4.7.3.8. *Restrictions on Implicit URI Grants*
- 4.7.3.9. *Cross-Profile Loopback Restriction*
- 4.12. *Using NFC and FeliCa*
- 5.2.3.9. *SMS OTP (One-Time Password) Protection*
- 5.4.3.9. *Enabling ECH (Encrypted Client Hello)*
- 5.4.3.10. *Certificate Transparency (CT) Compliance*
- 5.4.3.11. *Deprecation of usesCleartextTraffic and Permitting HTTP Communication*
- 5.5.3.15. *Transition to Contact Picker and Proper Handling of Contact Data*
- 5.6.3.10. *Per-App Android Keystore (Key) Limits*

Revised articles below

- 4.6.3.7. *Enhanced Safety of DCL (Dynamic Code Loading)*
- 5.1. *Creating Authentication (Password/Passkey) Screens*
- 5.4.1.1. *Communicating via HTTP*
- 5.4.1.2. *Communicating via HTTPS*
- 5.5.3.6. *Location Access*
- 5.5.3.7. *Local Network Access Permission*

Deleted the section below

- 4.1.3.7. Launching Activities Safely from Background

Note: For a detailed description of these revisions, see Section *About the Articles Corrected/Updated from the August 27, 2025 Edition*.

For the release of this new edition, we have updated the contents of this Guidebook based on your comments and suggestions.

Published by

Japan Smartphone Security Association (JSSEC), Technical Subcommittee, Secure Coding Working Group

Leader	Tsutomu Miyazaki	LAC Co., Ltd.
Members	Ryuta Nakagami	LAC Co., Ltd.
	Teruaki Honma	KDDI CORPORATION
	Harunobu Agematsu	KDDI CORPORATION

(In no particular order)

Authors of the August 27, 2025 Edition

Leader

Tsutomu Miyazaki, LAC Co., Ltd.

Members

Ryuta Nakagami	LAC Co., Ltd.
Akihiro Shiota	NTT DATA Corporation
Teruaki Honma	KDDI CORPORATION
Harunobu Agematsu	KDDI CORPORATION

Authors of January 29 2025 Edition

Leader

Tsutomu Miyazaki

LAC Co., Ltd.

Member

Ryuta Nakagami	Nuligen Security Co., Ltd.
Akihiro Shiota	NTT DATA Corporation
Teruaki Honma	KDDI CORPORATION
Harunobu Agematsu	KDDI CORPORATION

(In no particular order)

Authors of February 29 2024 Edition

Leader

Tsutomu Miyazaki

LAC Co., Ltd.

Member

Akihiro Shiota	NTT DATA Corporation
Teruaki Honma	KDDI CORPORATION
Harunobu Agematsu	KDDI CORPORATION
Naruhiko Ogasawara	SHIFT SECURITY
Yoshinori Saida	NEC Corporation
Toru Aoyagi	NEC Corporation

(In no particular order)

Authors of August 29 2022 Edition

Leader

Tsutomu Miyazaki

LAC Co., Ltd.

Member

Pantuhong Sorasiri	LAC Co., Ltd.
Akihiro Shiota	NTT DATA Corporation
Teruaki Honma	KDDI CORPORATION
Harunobu Agematsu	KDDI CORPORATION

(In no particular order)

Authors of January 17 2022 Edition

Leader

Tsutomu Miyazaki

LAC Co., Ltd.

Member

Nao Komatsu	LAC Co., Ltd.
Teruaki Honma	KDDI CORPORATION

(In no particular order)

Authors of October 19 2021 Edition

Leader

Tsutomu Miyazaki

LAC Co., Ltd.

Member

Nao Komatsu	LAC Co., Ltd.
Teruaki Honma	KDDI CORPORATION

(In no particular order)

Authors of November 1, 2020 Edition

Leader

Tsutomu Miyazaki

LAC Co., Ltd.

Member

Akihiro Shiota	NTT DATA Corporation
Teruaki Honma	KDDI CORPORATION
Saida Yoshinori	NEC Corporation

(In no particular order)

Authors of September 1 2019 Edition

Leader

Jun Ogiso

Sony Digital Network Applications, Inc.

Member

Toshimi Sawada	Software Research Associates, Inc.
Kohei Suzuki	Software Research Associates, Inc.
Akihiro Shiota	NTT DATA Corporation
Teruaki Honma	KDDI CORPORATION
Junki Hisamoto	Sony Digital Network Applications, Inc.
Nobuaki Yamaguchi	Sony Digital Network Applications, Inc.
Gaku Taniguchi	Tao Software, Inc.
Ito Takefumi	Nihon System Kaihatsu Co., Ltd.

(In no particular order)

Authors of September 1, 2018 Edition

Leader

Akira Ando

Sony Digital Network Applications, Inc.

Member

Toshimi Sawada	Software Research Associates, Inc.
Kohei Suzuki	Software Research Associates, Inc.
Teruaki Honma	KDDI CORPORATION
Jun Ogiso	Sony Digital Network Applications, Inc.
Junki Hisamoto	Sony Digital Network Applications, Inc.
Nobuaki Yamaguchi	Sony Digital Network Applications, Inc.
Shigeru Yatabe	Fomalhaut Techno Solutions

(In no particular order)

Authors of February 1, 2018 Edition

Leader

Akira Ando

Sony Digital Network Applications, Inc.

Member

Ken Okuyma	Android Security Japan
Eiji Hoshimoto	Software Research Associates, Inc.
Akihiro Shiota	NTT DATA Corporation
Shigenori Takei	NTT Software Corporation
Ikuya Fukumoto	Japan Computer Emergency Response Team Coordination Center (JPCERT/CC)
Mariko Yoshida	Sony Digital Network Applications, Inc.
Nobuaki Yamaguchi	Sony Digital Network Applications, Inc.
Jun Ogiso	Sony Digital Network Applications, Inc.
Junki Hisamoto	Sony Digital Network Applications, Inc.
Masahiro Kasahara	SoftBank Corp.
Ito Takefumi	Nihon System Kaihatsu Co., Ltd.
Shigeru Yatabe	Fomalhaut Techno Solutions

(In no particular order)

Authors of February 1, 2017 Edition

Leader

Ken Okuyama

Sony Digital Network Applications, Inc.

Member

Shigeharu Araki	Android Security Japan
Eiji Shimano	Android Security Japan
Akihiro Shiota	NTT DATA Corporation
Shigenori Takei	NTT Software Corporation
Ikuya Fukumoto	Software Research Associates, Inc.
Tomomi Ohuchi	Software Research Associates, Inc.
Yoichi Yamanoi	Software Research Associates, Inc.
Hidenori Yamaji	Sony Corporation
Akira Ando	Sony Digital Network Applications, Inc.
Jun Ogiso	Sony Digital Network Applications, Inc.
Masaru Matsunami	Sony Digital Network Applications, Inc.
Tetsuya Takahashi	SQUARE ENIX CO., LTD.
Gaku Taniguchi	Tao Software, Inc.

(In no particular order)

Authors of September 1, 2016 Edition

Leader

Masaru Matsunami

Sony Digital Network Applications, Inc.

Member

Shigeharu Araki	Android Security Japan
Shigenori Takei	NTT Software Corporation
Ikuya Fukumoto	Software Research Associates, Inc.
Tomomi Ohuchi	Software Research Associates, Inc.
Hidenori Yamaji	Sony Corporation
Akira Ando	Sony Digital Network Applications, Inc.
Jun Ogiso	Sony Digital Network Applications, Inc.
Ken Okuyama	Sony Digital Network Applications, Inc.
Mitake Ohtani	Sony Digital Network Applications, Inc.
Daisuke Mitsuzono	Nihon System Kaihatsu Co., Ltd.
Eiji Shimano	Tao Software, Inc.
Gaku Taniguchi	Tao Software, Inc.

(In no particular order)

Authors of February 1, 2016 Edition

Leader

Masaru Matsunami

Sony Digital Network Applications, Inc.

Member

Masaomi Adachi	Android Security Japan
Tohru Ohzono	Cisco Systems, Inc.
Shigenori Takei	NTT Software Corporation
Masahiro Kasahara	SoftBank Mobile Corp.
Eiji Hoshimoto	Software Research Associates, Inc.
Ikuya Fukumono	Software Research Associates, Inc.
Akira Ando	Sony Digital Network Applications, Inc.
Ken Okuyama	Sony Digital Network Applications, Inc.
Mitake Ohtani	Sony Digital Network Applications, Inc.
Muneaki Nishimura	Sony Digital Network Applications, Inc.
Setsuko Kaji	Sony Digital Network Applications, Inc.
Taeko Ito	Sony Digital Network Applications, Inc.
Hidenori Yamaji	Sony Mobile Communications Inc.
Eiji Shimano	Tao Software, Inc.
Gaku Taniguchi	Tao Software, Inc.

(In no particular order)

Authors of June 1, 2015 Edition

Leader

Masaru Matsunami

Sony Digital Network Applications, Inc.

Member

Tohru Ohzono	Cisco Systems, Inc.
Akio Kondo	BRILLIANTSERVICE co., Ltd.
Kazuma Mitake	BRILLIANTSERVICE co., Ltd.
Kyosuke Imanishi	BRILLIANTSERVICE co., Ltd.
Masato Shintani	BRILLIANTSERVICE co., Ltd.
Naohiko Shimura	BRILLIANTSERVICE co., Ltd.
Ryuji Fujita	BRILLIANTSERVICE co., Ltd.
Shohei Hara	BRILLIANTSERVICE co., Ltd.
Tomoyuki Fujisawa	BRILLIANTSERVICE co., Ltd.
Yutaka Kawahara	BRILLIANTSERVICE co., Ltd.
Shigeru Yatabe	Fomalhaut Techno Solutions
Naonobu Yatsukawa	Nihon Unisys, Ltd.
Shigenori Takei	NTT Software Corporation
Masahiro Kasahara	SoftBank Mobile Corp.
Eiji Hoshimoto	Software Research Associates, Inc.
Akira Ando	Sony Digital Network Applications, Inc.
Ken Okuyama	Sony Digital Network Applications, Inc.
Muneaki Nishimura	Sony Digital Network Applications, Inc.
Eiji Shimano	Tao Software, Inc.
Gaku Taniguchi	Tao Software, Inc.

(In no particular order)

Authors of July 1, 2014 English Edition

Leader

Masaru Matsunami

Sony Digital Network Applications, Inc.

Member

Tohru Ohzono	Cisco Systems, Inc.
Shigeru Yatabe	Fomalhaut Techno Solutions
Keisuke Takemori	KDDI CORPORATION
Takamasa Isohara	KDDI CORPORATION
Naonobu Yatsukawa	Nihon Unisys, Ltd.
Shigenori Takei	NTT Software Corporation
Masahiro Kasahara	SoftBank Mobile Corp.
Eiji Hoshimoto	Software Research Associates, Inc.
Tsutomu Kumazawa	Software Research Associates, Inc.
Akira Ando	Sony Digital Network Applications, Inc.
Ken Okuyama	Sony Digital Network Applications, Inc.
Setsuko Kaji	Sony Digital Network Applications, Inc.
Taeko Ito	Sony Digital Network Applications, Inc.
Yoshinori Kataoka	Sony Digital Network Applications, Inc.
Eiji Shimano	Tao Software, Inc.
Gaku Taniguchi	Tao Software, Inc.
Michiyoshi Sato	Tokyo System House Co., Ltd.

(In no particular order)

Authors of April 1, 2014 English Edition

Leader

Masaru Matsunami

Sony Digital Network Applications, Inc.

Member

Tomoyuki Hasegawa	Android Security Japan
Mayumi Nishiyama	BJIT Inc.
Tohru Ohzono	Cisco Systems, Inc.
Masaki Kubo	Japan Computer Emergency Response Team Coordination Center (JPCERT/CC)
Daniel Burrowes	Kobe Digital Labo Inc.
Zachary Mathis	Kobe Digital Labo Inc.
Renta Futamura	NextGen, Inc.
Naonobu Yatsukawa	Nihon Unisys, Ltd.
Shigenori Takei	NTT Software Corporation
Ikuya Fukumono	Software Research Associates, Inc.
Tsutomu Kumazawa	Software Research Associates, Inc.
Akira Ando	Sony Digital Network Applications, Inc.
Hiroko Nakajima	Sony Digital Network Applications, Inc.
Ken Okuyama	Sony Digital Network Applications, Inc.
Satoshi Fujimura	Sony Digital Network Applications, Inc.
Setsuko Kaji	Sony Digital Network Applications, Inc.
Taeko Ito	Sony Digital Network Applications, Inc.
Yoshinori Kataoka	Sony Digital Network Applications, Inc.
Hidenori Yamaji	Sony Mobile Communications Inc.
Takuya Nishibayashi	Sony Mobile Communications Inc.
Koji Isoda	Symantec Japan, Inc.
Gaku Taniguchi	Tao Software, Inc.
Michiyoshi Sato	Tokyo System House Co., Ltd.

(In no particular order)

Authors of April 1, 2013 Japanese Edition

Leader

Masaru Matsunami

Sony Digital Network Applications, Inc.

Member

Masaomi Adachi	Android Security Japan
Tomoyuki Hasegawa	Android Security Japan
Yuki Abe	Software Research Associates, Inc.
Tomomi Oouchi	Software Research Associates, Inc.
Tsutomu Kumazawa	Software Research Associates, Inc.
Toshimi Sawada	Software Research Associates, Inc.
Kiyoshi Hata	Software Research Associates, Inc.
Youichi Higa	Software Research Associates, Inc.
Yuu Fukui	Software Research Associates, Inc.
Ikuya Fukumoto	Software Research Associates, Inc.
Eiji Hoshimoto	Software Research Associates, Inc.
Shun Yokoi	Software Research Associates, Inc.
Takakazu Yoshizawa	Software Research Associates, Inc.
Takeshi Fujiwara	NRI SecureTechnologies, Ltd.
Shigenori Takei	NTT Software Corporation
Masaki Kubo	Japan Computer Emergency Response Team Coordination Center(JPCERT/CC)
Hiroshi Kumagai	Japan Computer Emergency Response Team Coordination Center(JPCERT/CC)
Yozo Toda	Japan Computer Emergency Response Team Coordination Center(JPCERT/CC)
Tohru Ohzono	Cisco Systems, Inc.
Toru Asano	Sony Digital Network Applications, Inc.
Akira Ando	Sony Digital Network Applications, Inc.
Ryohji Ikebe	Sony Digital Network Applications, Inc.
Jun Ogiso	Sony Digital Network Applications, Inc.
Ken Okuyama	Sony Digital Network Applications, Inc.
Yoshinori Kataoka	Sony Digital Network Applications, Inc.
Muneaki Nishimura	Sony Digital Network Applications, Inc.
Koji Furusawa	Sony Digital Network Applications, Inc.
Kenji Yamaoka	Sony Digital Network Applications, Inc.
Gaku Taniguchi	Tao Software, Inc.
Naonobu Yatsukawa	Nihon Unisys, Ltd.
Shigeru Yatabe	Fomalhaut Techno Solutions

(In no particular order)

Authors of November 1, 2012 Japanese Edition

Leader

Masaru Matsunami

Sony Digital Network Applications, Inc.

Member

Katsuhiko Sato	Android Security Japan
Nakaguchi Akihiko	Android Security Japan
Tomomi Oouchi	Software Research Associates, Inc.
Naoyuki Ohira	Software Research Associates, Inc.
Tsutomu Kumazawa	Software Research Associates, Inc.
Miki Sekikawa	Software Research Associates, Inc.
Seigo Nakano	Software Research Associates, Inc.
Youichi Higa	Software Research Associates, Inc.
Ikuya Fukumoto	Software Research Associates, Inc.
Eiji Hoshimoto	Software Research Associates, Inc.
Shoichi Yasuda	Software Research Associates, Inc.
Tadayuki Yahiro	Software Research Associates, Inc.
Takakazu Yoshizawa	Software Research Associates, Inc.
Shigenori Takei	NTT Software Corporation
Keisuke Takemori	KDDI CORPORATION
Masaki Kubo	Japan Computer Emergency Response Team Coordination Center(JPCERT/CC)
Hiroshi Kumagai	Japan Computer Emergency Response Team Coordination Center(JPCERT/CC)
Yozo Toda	Japan Computer Emergency Response Team Coordination Center(JPCERT/CC)
Tohru Ohzono	Cisco Systems, Inc.
Toru Asano	Sony Digital Network Applications, Inc.
Akira Ando	Sony Digital Network Applications, Inc.
Ryohji Ikebe	Sony Digital Network Applications, Inc.
Shigeru Ichikawa	Sony Digital Network Applications, Inc.
Mitake Ohtani	Sony Digital Network Applications, Inc.
Jun Ogiso	Sony Digital Network Applications, Inc.
Ken Okuyama	Sony Digital Network Applications, Inc.
Yoshinori Kataoka	Sony Digital Network Applications, Inc.
Ikue Sato	Sony Digital Network Applications, Inc.
Muneaki Nishimura	Sony Digital Network Applications, Inc.
Kazuo Yamaoka	Sony Digital Network Applications, Inc.
Takeru Kikkawa	Sony Digital Network Applications, Inc.
Gaku Taniguchi	Tao Software, Inc.
Eiji Shimano	Tao Software, Inc.
Hisao Kitamura	Tao Software, Inc.
Takao Yamakawa	Japan Online Game Association
Masaki Ishihara	Nihon System Kaihatsu Co., Ltd.
Yasuaki Mori	Nihon System Kaihatsu Co., Ltd.
Naonobu Yatsukawa	Nihon Unisys, Ltd.
Shigeru Yatabe	Fomalhaut Techno Solutions
Shigeki Fujii	UNIADEX, Ltd.

(In no particular order)

Authors of June 1, 2012 Japanese Edition

Leader

Masaru Matsunami

Sony Digital Network Applications, Inc.

Member

Katsuhiko Sato	Android Security Japan
Tomomi Oouchi	Software Research Associates, Inc.
Youichi Higa	Software Research Associates, Inc.
Eiji Hoshimoto	Software Research Associates, Inc.
Shigenori Takei	NTT Software Corporation
Masaaki Chida	GREE, Inc.
Masaki Kubo	Japan Computer Emergency Response Team Coordination Center(JPCERT/CC)
Hiroshi Kumagai	Japan Computer Emergency Response Team Coordination Center(JPCERT/CC)
Yozo Toda	Japan Computer Emergency Response Team Coordination Center(JPCERT/CC)
Tohru Ohzono	Cisco Systems, Inc.
Yoichi Taguchi	System House. ING Co., Ltd.
Masahiko Sakamoto	Secure Sky Technology, Inc.
Akira Ando	Sony Digital Network Applications, Inc.
Shigeru Ichikawa	Sony Digital Network Applications, Inc.
Ken Okuyama	Sony Digital Network Applications, Inc.
Shigeru Ichikawa	Sony Digital Network Applications, Inc.
Ken Okuyama	Sony Digital Network Applications, Inc.
Ikue Sato	Sony Digital Network Applications, Inc.
Muneaki Nishimura	Sony Digital Network Applications, Inc.
Kazuo Yamaoka	Sony Digital Network Applications, Inc.
Gaku Taniguchi	Tao Software, Inc.
Eiji Shimano	Tao Software, Inc.
Hisao Kitamura	Tao Software, Inc.
Michiyoshi Sato	Tokyo System House Co., Ltd.
Masakazu Hattori	Trend Micro Incorporated.
Naonobu Yatsukawa	Nihon Unisys, Ltd.
Shigeru Yatabe	Fomalhaut Techno Solutions
Shigeki Fujii	UNIADEX, Ltd.

(In no particular order)