JAPAN SMARTPHONE SECURITY ASSOCIATION

Android アプリのセキュア設計・セキュアコーディングガイド

2026 年 9 月 16 日版

一般社団法人日本スマートフォンセキュリティ協会（JSSEC）

セキュアコーディング WG

文書管理番号：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。

目次

1 はじめに 2
1.1 スマートフォンを安心して利用出来る社会へ . . . 2
1.2 常にベータ版でタイムリーなフィードバックを . . . 3
1.3 本文書の利用許諾 . . . 3
1.4 2025 年 8 月 27 日版からの訂正記事について . . . 3

2 ガイド文書の構成 5
2.1 開発者コンテキスト . . . 5
2.2 サンプルコード、ルールブック、アドバンスト . . . 5
2.3 ガイド文書のスコープ . . . 7
2.4 Android セキュアコーディング関連書籍の紹介 . . . 8
2.5 サンプルコードの Android Studio への取り込み手順 . . . 9

3 セキュア設計・セキュアコーディングの基礎知識 14
3.1 Android アプリのセキュリティ . . . 14
3.2 入力データの安全性を確認する . . . 24

4 安全にテクノロジーを活用する 26
4.1 Activity を作る・利用する . . . 26
4.2 Broadcast を受信する・送信する . . . 93
4.3 Content Provider を作る・利用する . . . 135
4.4 Service を作る・利用する . . . 206
4.5 SQLite を使う . . . 264
4.6 ファイルを扱う . . . 283
4.7 Intent を利用する . . . 335
4.8 LogCat にログ出力する . . . 348
4.9 WebView を使う . . . 359
4.10 Notification を使用する . . . 378
4.11 共有メモリを使用する . . . 398
4.12 NFC・FeliCa を利用する . . . 427

5 セキュリティ機能の使い方 435
5.1 認証（パスワード・パスキー）画面を作る . . . 435
5.2 Permission と Protection Level . . . 443

5.3 Account Manager に独自アカウントを追加する . . . 484
5.4 HTTPS で通信する . . . 506
5.5 プライバシー情報を扱う . . . 544
5.6 暗号技術を利用する . . . 600
5.7 生体認証機能を利用する . . . 637

6 難しい問題 657
6.1 Clipboard から情報漏洩する危険性 . . . 657

更新履歴 670
制作 . . . 677
2025 年 8 月 27 日版制作者 . . . 679
2025 年 1 月 29 日版制作者 . . . 680
2024 年 2 月 29 日版制作者 . . . 681
2022 年 8 月 29 日版制作者 . . . 682
2022 年 1 月 17 日版制作者 . . . 683
2021 年 10 月 19 日版制作者 . . . 684
2020 年 11 月 1 日版制作者 . . . 685
2019 年 9 月 1 日版制作者 . . . 686
2018 年 9 月 1 日版制作者 . . . 687
2018 年 2 月 1 日版制作者 . . . 688
2017 年 2 月 1 日版制作者 . . . 689
2016 年 9 月 1 日版制作者 . . . 690
2016 年 2 月 1 日版制作者 . . . 691
2015 年 6 月 1 日版制作者 . . . 692
2014 年 7 月 1 日版制作者 . . . 693
2013 年 4 月 1 日版制作者 . . . 694
2012 年 11 月 1 日版制作者 . . . 695
2012 年 6 月 1 日版制作者 . . . 696

2026 年 **9** 月 **16** 日版

一般社団法人日本スマートフォンセキュリティ協会（**JSSEC**）

セキュアコーディング **WG**

※ 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。

1 はじめに

1.1 スマートフォンを安心して利用出来る社会へ

本ガイドは Android アプリケーション開発者向けのセキュア設計、セキュアコーディングのノウハウをまとめた Tips 集です。できるだけ多くの Android アプリケーション開発者に活用していただきたく思い、ここに公開いたします。

昨今、スマートフォン市場は急拡大しており、さらにその勢いは増すばかりです。スマートフォン市場の急拡大は多種多彩なアプリケーション群によってもたらされています。従来の携帯電話ではセキュリティ制約によって利用できなかったさまざまな携帯電話の重要な機能がスマートフォンアプリケーションには開放され、従来の携帯電話では実現できなかった多種多彩なアプリケーション群がスマートフォンの魅力を引き立てています。

スマートフォンのアプリケーション開発者にはそれ相応の責任が生じています。従来の携帯電話ではあらかじめ課せられたセキュリティ制約によって、セキュリティについてあまり意識せずに開発したアプリケーションであっても比較的安全性が保たれていました。スマートフォンでは前述のとおり、携帯電話の重要な機能がアプリケーション開発者に開放されているため、アプリケーション開発者がセキュリティを意識して設計、コーディングをしなければ、スマートフォン利用者の個人情報が漏洩したり、料金の発生する携帯電話機能をマルウェアに悪用されたりといった被害が生じます。

Android スマートフォンは iPhone に比べると、アプリケーション開発者のセキュリティへの配慮がより多く求められます。iPhone に比べ Android スマートフォンはアプリケーション開発者に開放された携帯電話機能が多く、App Store に比べ Google Play（旧 Android Market）は無審査でアプリケーション公開ができるなど、アプリケーションのセキュリティがほぼ全面的にアプリケーション開発者に任されているためです。

スマートフォン市場の急拡大にともない、様々な分野のソフトウェア技術者が一気にスマートフォンアプリケーション開発市場に流れ込んできており、スマートフォン特有のセキュリティを考慮したセキュア設計、セキュアコーディングのノウハウ集約、共有が急務となっています。

このような状況を踏まえ、一般社団法人日本スマートフォンセキュリティ協会はセキュアコーディング WG を立ち上げ、Android アプリケーションのセキュア設計、セキュアコーディングのノウハウを集めて、公開することにいたしました。それがこのガイド文書です。多くの Android アプリケーション開発者にセキュア設計、セキュアコーディングのノウハウを知っていただき、アプリケーション開発に活かしていただくことで、市場にリリースされる多くの Android アプリケーションのセキュリティを高めることを狙っています。その結果、安心、安全なスマートフォン社会づくりに貢献したいと考えています。

1.2 常にベータ版でタイムリーなフィードバックを

私たち JSSEC セキュアコーディング WG はこのガイド文書の内容について、できるだけ間違いがないように心がけておりますが、その正しさを保証するものではありません。私たちはタイムリーにノウハウを公開し共有していくことが第一と考え、最新かつその時点で正しいと思われることをできるだけ記載・公開し、間違いがあればフィードバックを頂いて常に正しい情報に更新し、タイムリーに提供するよう心がける、いわゆる常にベータ版というアプローチをとっています。このアプローチはこのガイド文書をご利用いただく多くの Android アプリケーション開発者のみなさまにとって有意義であると私たちは信じています。

このガイド文書とサンプルコードの最新版はいつでも下記 URL から入手できます。

- https://www.jssec.org/dl/android_securecoding/ ガイド文書
- https://www.jssec.org/dl/android_securecoding.zip サンプルコード一式

1.3 本文書の利用許諾

このガイド文書のご利用に際しては次の 2 つの注意事項に同意いただく必要がございます。

1. このガイド文書には間違いが含まれている可能性があります。ご自身の責任のもとでご利用ください。
2. このガイド文書に含まれる間違いを見つけた場合には、下記連絡先までメールにてご連絡ください。ただしお返事することや修正をお約束するものではありませんのでご了承ください。

一般社団法人 日本スマートフォンセキュリティ協会

問い合わせ窓口

URL：https://www.jssec.org/contact

1.4 2025 年 8 月 27 日版からの訂正記事について

本節では、前版の記事について事実関係と照らし合わせることで判明した訂正事項を一覧にして掲載しています。各訂正記事は、執筆者による継続的な調査結果だけでなく読者の方々の貴重なご指摘を広く取り入れたものです。特に、いただいたご指摘は、本改訂版をより実践に即したガイドとして高い完成度を得るための最も重要な糧となっています。

前版を元にアプリケーション開発を進めていた読者は、以下の訂正記事一覧に特に目を通していただきますようお願いいたします。なお、ここで掲げる項目には、誤植の修正、構成の変更、単なる表現上の改善は含みません。

本ガイドに対するコメントは、今後もお気軽にお寄せくださいますようよろしくお願いいたします。

訂正記事一覧

表 1.4.1　訂正記事一覧

2025 年 8 月 27 日版の修正個所	本改訂版の訂正記事	訂正の要旨
（該当なし）	4.1.3.7. バックグラウンドアクティビティ起動（*BAL*）の制限強化	バックグラウンドからのアクティビティ起動（BAL）の制限強化について追記しました。
（該当なし）	4.2.3.10. *Bluetooth* ボンドの損失に対する自律的な再ペア設定	Bluetooth ボンドの損失に対する自律的な再ペア設定について追記しました。
（該当なし）	4.3.3.2. 連絡先プロバイダ（*CP2*）の厳格化	連絡先プロバイダ（CP2）アクセスの厳格化について追記しました。
（該当なし）	4.4.3.5. バックグラウンドオーディオの制限強化	バックグラウンドオーディオの制限強化について追記しました。
（該当なし）	4.7.3.8. 暗黙的な *URI* 権限付与の制限	暗黙的な URI 権限付与の制限について追記しました。
（該当なし）	4.7.3.9. プロファイル間のループバック制限	プロファイル間のループバック制限について追記しました。
（該当なし）	4.12. *NFC・FeliCa* を利用する	NFC・FeliCa を利用するについて追記しました。
（該当なし）	5.2.3.9. *SMS OTP*（ワンタイムパスワード）の保護強化	SMS OTP（ワンタイムパスワード）の保護強化について追記しました。
（該当なし）	5.4.3.9. *ECH*（*Encrypted Client Hello*）の有効化	ECH（Encrypted Client Hello）の有効化について追記しました。
（該当なし）	5.4.3.10. 証明書の透明性（*CT*）への対応	証明書の透明性（CT）の自動検証について追記しました。
（該当なし）	5.4.3.11. *usesCleartextTraffic* の非推奨化と *HTTP* 通信の許可	usesCleartextTraffic の非推奨化について追記しました。
（該当なし）	5.6.3.10. アプリごとの *Android Keystore*（鍵）上限	アプリごとのキーストア（鍵）上限について追記しました。
• 4.6.3.7. DCL (Dynamic Code Loading) の安全性の強化	4.6.3.7. *DCL (Dynamic Code Loading)* の安全性の強化	ネイティブ DCL（動的コード読み込み）の安全化について記載内容の見直しをしました。
• 5.1. パスワード入力画面を作る	5.1. 認証（パスワード・パスキー）画面を作る	物理デバイスからパスワードを非表示にする設定について追記しました。
• 5.4.1.1. HTTP 通信する	5.4.1.1. *HTTP* 通信する	HTTP 通信における usesCleartextTraffic の非推奨化対応について記載内容の見直しをしました。
• 5.4.1.2. HTTPS 通信する	5.4.1.2. *HTTPS* 通信する	HTTPS 通信における ECH や証明書の透明性（CT）自動検証について記載内容の見直しをしました。
• 5.5.3.6. 位置情報アクセスについて	5.5.3.6. 位置情報アクセスについて	位置情報アクセスにおける位置情報ボタンの導入について記載内容の見直しをしました。
• 5.5.3.7. ローカル ネットワークへのアクセス権	5.5.3.7. ローカル ネットワークへのアクセス権	ローカルネットワーク権限の必須化について記載内容の見直しをしました。
• 4.1.3.7. 安全なバックグラウンド アクティビティの起動	（該当なし）	安全なバックグラウンドアクティビティの起動に関する記事を削除しました。

2 ガイド文書の構成

2.1 開発者コンテキスト

セキュアコーディング系のガイド文書は「こういうコーディングは危ない、だからこのようにコーディングすべき」といった内容で構成されることが多いのですが、このような構成はすでにコーディングされたソースコードをレビューするときには役立つ反面、これから開発者がコーディングしようというときには、どの記事を読んだらよいのか分かりにくいという問題があります。

このガイド文書では、開発者がいま何をしようとしているか？ という開発者コンテキストに着目し、開発者コンテキストに合わせた切り口の記事を用意する方針をとっています。たとえば「Activity を作る・利用する」や「SQLite を使う」という開発者が行うであろう作業単位ごとに記事を用意しています。

開発者コンテキストに合わせて記事を用意することにより、開発者は必要な記事を見つけやすく、業務にすぐ役立つようになると考えています。

2.2 サンプルコード、ルールブック、アドバンスト

それぞれの記事はサンプルコード、ルールブック、アドバンストの 3 つのセクションで構成されています。お急ぎの方はサンプルコードとルールブックをご覧ください。ある程度再利用可能なパターンに落とし込んだ内容になっています。サンプルコードセクションとルールブックセクションに収まらない課題をお持ちの方はアドバンストをご覧ください。個別課題の解決方法を検討するための考慮材料を記載しています。

サンプルコードおよび記事の内容は、特別な記述がない限り Android 7.0（API Level 24）以降を対象としています。Android 7.0（API Level 24）より前のバージョンについては動作確認を行っておらず、対策として効果がない場合もありますのでご注意ください。また、対象範囲内のバージョンであっても、組み込んだ端末で動作をご確認の上、ご自身の責任でご利用ください。

また、Android 14 および Android 15 以降では、インストール可能なアプリの targetSdkVersion に最低要件が設けられています。これは、マルウェアが古い API レベルをターゲットにして新しいセキュリティ保護を回避することを防ぐためであり、そのようなマルウェアを一律に排除することを目的としています。

Android バージョンごとのインストール可能な targetSdkVersion は以下の通りです。

Android バージョン	インストール可能な targetSdkVersion
Android 7.0 - 13	制約なし
Android 14	targetSdkVersion 23 以上
Android 15	targetSdkVersion 24 以上

2.2.1 サンプルコード

サンプルコードセクションでは、その記事がテーマとする開発者コンテキストにおいて基本的なお手本となるサンプルコードを掲載しています。複数のパターンがある場合はその分類方法とそれぞれのパターンのサンプルコードを用意しています。解説においては簡潔さを心がけており、セキュリティ上考慮すべきポイントを本文中で「ポイント：」部分に番号付き箇条書きで記載し、その箇条書き番号 N に対応するサンプルコードにも「★ポイント **N** ★」と記載しコメントで解説しています。一つのポイントがサンプルコード上では複数個所に対応する場合があることにご注意ください。このようにセキュリティを考慮すべき個所はソースコード全体に対して僅かな量ですが、それらの個所は点在します。セキュリティの考慮が必要な個所を見渡すことができるように、サンプルコードはクラス単位でまるごと掲載するようにしています。

このガイド文書で掲載しているサンプルコードは一部です。すべてのサンプルコードをまとめた圧縮ファイルも下記の URL に公開しています。Apache License, Version 2.0 で公開していますので、自由にサンプルコードをコピー&ペーストしてご利用いただけます。ただしエラー処理についてはサンプルコードが長くなり過ぎないように最小限にしていますのでご注意ください。

- https://www.jssec.org/dl/android_securecoding/ ガイド文書
- https://www.jssec.org/dl/android_securecoding.zip サンプルコード一式

サンプルコードに添付する Projects/keystore ファイルは APK 署名用の開発者鍵を含んだキーストアファイルです。パスワードは「android」です。自社限定系のサンプルコードを APK 署名する際にご利用ください。

デバッグ用にキーストアファイル debug.keystore を用意しているので、Android Studio で開発する場合は、Android Studio の個別のプロジェクトで設定しておくと、自社限定系のサンプルコードの動作確認に便利です。また、複数の APK から成るサンプルコードにおいて、各 APK 間の連携動作を確認するためには、各々の AndroidManifest.xml 内の android:debuggable の設定を合わせる必要があります。Android Studio から APK をインストールする場合は、明示的に設定が無ければ自動的に android:debuggable=”true” になります。

サンプルコードおよびキーストアファイルを Android Studio に取り込む方法については「2.5. サンプルコードの *Android Studio* への取り込み手順」をご参照ください。

2.2.2 ルールブック

ルールブックセクションでは、その記事がテーマとする開発者コンテキストにおいて、セキュリティ観点から守るべきルールや考慮事項を掲載しています。ルールブックセクションの冒頭にはそのセクションで扱っているルールを表形式で一覧表示し、「必須」または「推奨」のレベル分けをしています。ルールには肯定文または否定文の 2 種類がありますので、必須の肯定文は「やらなきゃだめ」、推奨の肯定文は「やったほうがよい」、必須の否定文は「やったらだめ」、推奨の否定文は「やらないほうがよい」といったレベル感で表現しています。もちろんこのレベル分けは執筆者の主観に基づくものですので、参考程度としてお取扱いください。

サンプルコードセクションに掲載されているサンプルコードはこれらのルールや考慮事項が反映されたものとなってい

ますが、その詳しい説明はルールブックセクションに記載されています。また、サンプルコードセクションでは扱っていないルールや考慮事項についてもルールブックセクションでは扱っています。

2.2.3 アドバンスト

アドバンストセクションでは、その記事がテーマとする開発者コンテキストにおいて、サンプルコードセクションやルールブックセクションで説明できなかった、しかし注意を要する事項について記載しています。その記事がテーマとする開発者コンテキストにまつわる、コラム的な話題や Android OS の限界に関する話題など、サンプルコードセクションやルールブックセクションの内容で解決できなかった個別課題の解決方法を検討するための考慮材料として役立てることができます。

開発者のみなさんは常に多忙です。開発者の多くは、Android の深遠なるセキュリティの構造について深く理解することよりも、ある程度の Android セキュリティの知識を持って、迅速にかつ安全な Android アプリケーションをどんどん生産することが求められます。一方、セキュリティ設計が重要なアプリケーションもあります。このようなアプリケーションの開発者は Android のセキュリティについて深く理解している必要があります。

このようにスピード重視の開発者とセキュリティ重視の開発者の両方を支援するために、このガイド文書のすべての記事はサンプルコード、ルールブック、アドバンストの 3 つのセクションに分けて記述しています。サンプルコードとルールブックセクションは「そういうことがしたければ、これをしておけば安全ですよ」といった一般化できる内容が書いてあり、可能な限りソースコードのコピー&ペーストで自動的に安全なコーディングができることを狙っています。アドバンストセクションは「こんなときはこういう問題があって、こういう考え方をするとよい」といった考えるための材料が書いてあり、開発者が取り組んでいる個別のアプリケーションで最適なセキュア設計、セキュアコーディングを検討できることを狙っています。

2.3 ガイド文書のスコープ

このガイド文書は一般の Android アプリケーション開発者に必要なセキュリティ Tips を集めることを目的としています。そのため主にマーケットで配布される Android アプリケーションの開発におけるセキュリティ Tips（下図の「アプリのセキュリティ」）が主なスコープとなっています。

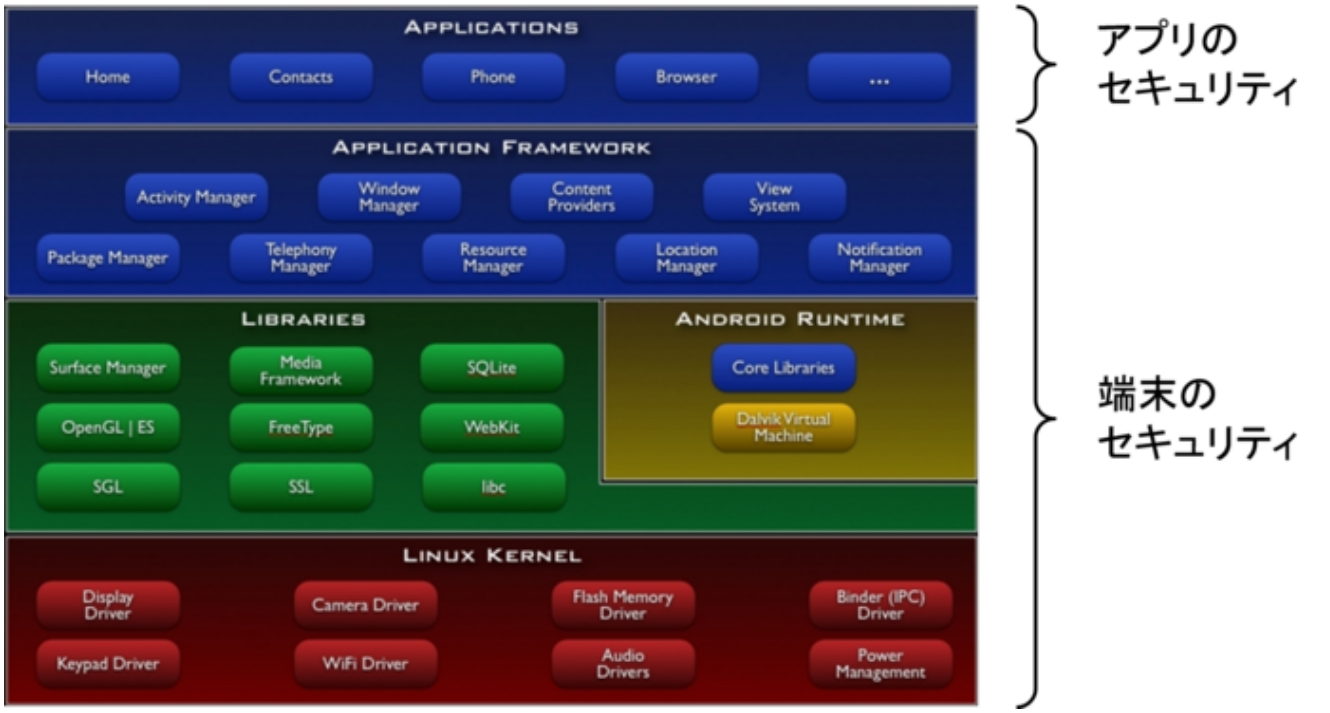

図 2.3.1　Android プラットフォームの主なコンポーネント

Android OS 層以下の Android 端末実装に関するセキュリティ（上図の「端末のセキュリティ」）はスコープ外です。また Android 端末にユーザーがインストールする一般の Android アプリケーションと、Android 端末メーカーがプレインストールする Android アプリケーションでは気を付けるべきセキュリティの観点で異なるところがありますが、特に現行版においては前者のみを扱っており、後者については扱っていません。現行版では Java により実装する Tips だけを記載しておりますが、JNI 実装についても今後の版で記載していく予定です。

root 権限が奪取される脅威についても今のところ扱っていません。基本的には root 権限が奪われていないセキュアな Android 端末を前提とし、Android OS のセキュリティモデルを活用したセキュリティ Tips をまとめています。なお、資産と脅威の扱いについては「3.1.3. 資産分類と保護施策」にて詳しく説明しておりますので、合わせてご確認ください。

2.4 Android セキュアコーディング関連書籍の紹介

このガイド文書では Android セキュアコーディングのすべてを扱うことはとてもできないので、下記で紹介する書籍を併用することをお勧めします。

Android Security 安全なアプリケーションを作成するために

著者：タオソフトウェア株式会社　ISBN978-4-8443-3134-6

https://www.amazon.co.jp/dp/4844331345/

Java セキュアコーディングスタンダード CERT/ Oracle 版

著者：Fred Long, Dhruv Mohindra, Robert C. Seacord, Dean F. Sutherland, David Svoboda

監修：歌代和正　翻訳：久保正樹, 戸田洋三　ISBN978-4-04-886070-3

https://www.amazon.co.jp/dp/4048860704/

OWASP Mobile Application Security Verification Standard (MASVS)

著者: The OWASP Foundation

https://github.com/OWASP/owasp-masvs/

OWASP Top 10

著者: The OWASP Foundation

https://owasp.org/Top10/

モバイルアプリケーション開発 10 大チェックポイント 2023

著者: JSSEC 技術部会 モバイルアプリケーション開発 10 大チェックポイント 2023 選定委員

https://www.jssec.org/mobile-apps-10checkpoint2023

2.5 サンプルコードの Android Studio への取り込み手順

サンプルコードの Android Studio への取り込み手順を説明します。サンプルコードは目的ごとに複数のプロジェクトにわかれています。これらのプロジェクトを取り込む方法を「2.5.1. サンプルプロジェクトを取り込む」に示します。

2.5.1 サンプルプロジェクトを取り込む

2.5.1.1 サンプルコードをダウンロードする

「2.2.1. サンプルコード」で紹介した URL よりサンプルコードを取得します。

2.5.1.2 サンプルコードを展開する

Zip で圧縮されたサンプルコードを右クリックし、表示されたメニューの” すべて展開” をクリックします。

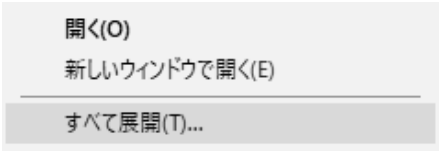

図 2.5.1　サンプルコードを展開する

2.5.1.3 展開先を指定する

ここでは”C:\android_securecoding” という名前でワークスペースを作成します。そのため、”C:\ ” を指定し” 展開” ボタンをクリックします。

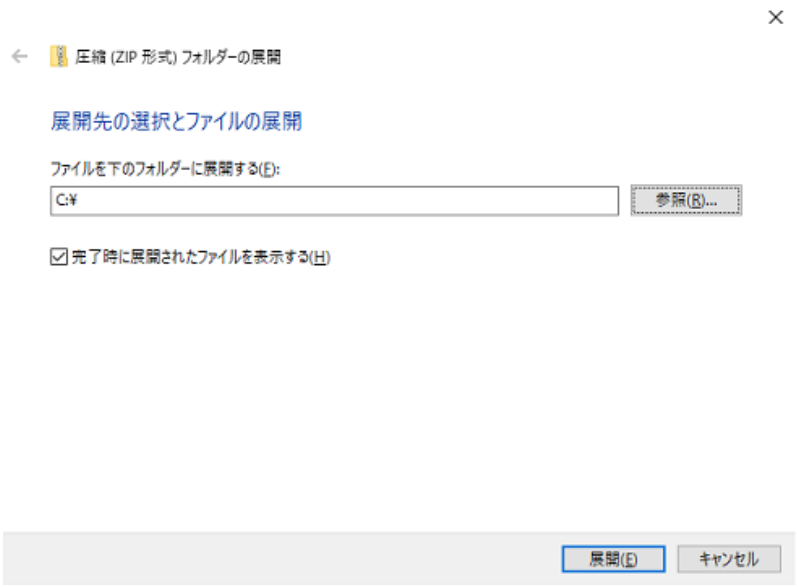

図 2.5.2　展開先を指定する

” 展開” ボタンをクリックすると”C:\” 直下に”android_securecoding” というフォルダが作成されます。

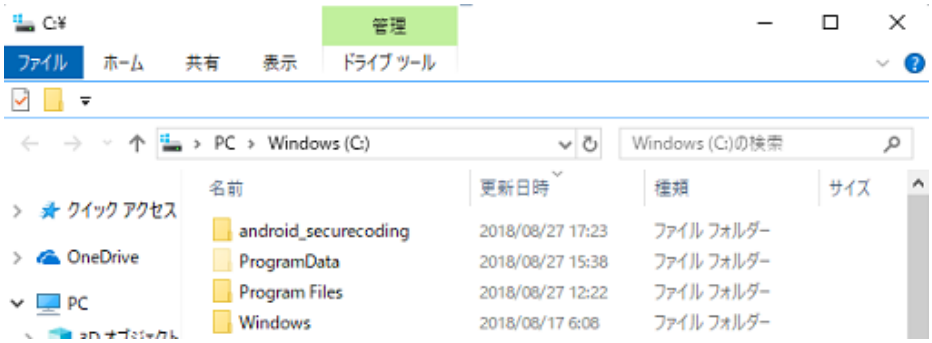

図 2.5.3　android_securecoding フォルダ

”android_securecoding” フォルダの中にはサンプルコードが含まれています。

例えば、「4.1. *Activity* を作る・利用する」の「4.1.1.4. 自社限定 *Activity* を作る・利用する」においてサンプルコードを参照したい場合は以下をご覧ください。

```
android_securecoding
    ┗ Create Use Activity
        ┗ Activity InhouseActivity
```

以上のように、”android_securecoding” フォルダ配下は、節ごとに「サンプルコードのプロジェクト」が配置された構成となります。

2.5.1.4 サンプルコードをビルドする

スタートメニューやデスクトップアイコンなどから Android Studio を起動します。

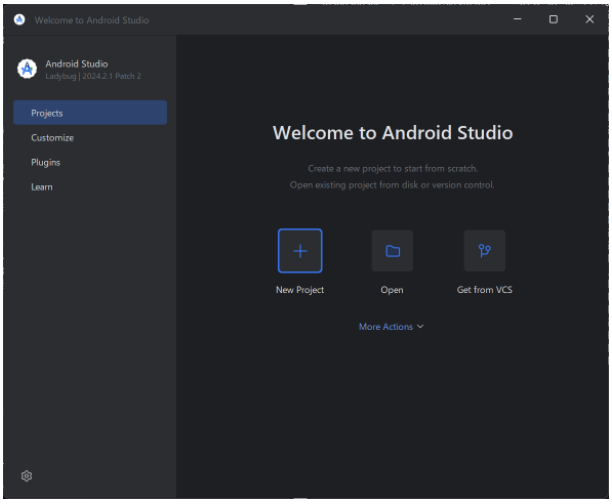

図 2.5.4　Android Studio を起動

Open をクリックします。

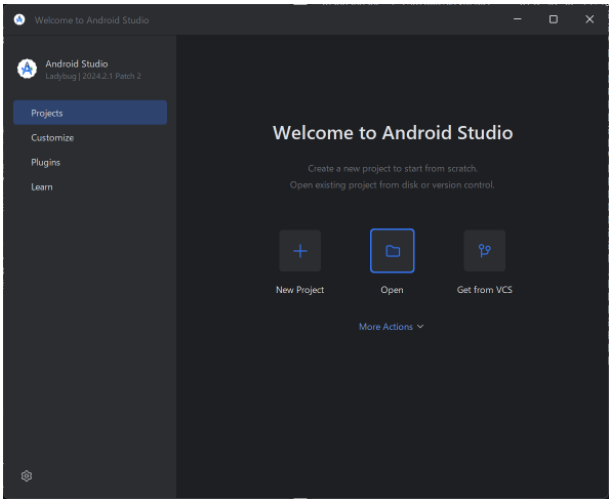

図 2.5.5　Open をクリック

オープンするプロジェクトフォルダを選択します。

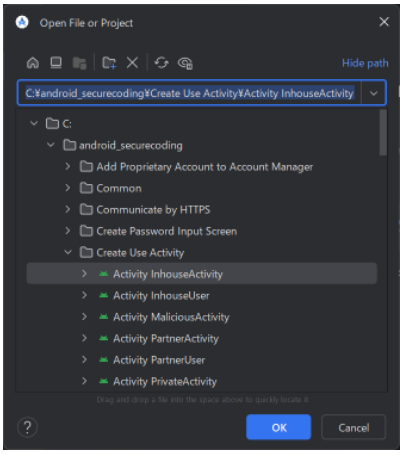

図 2.5.6　プロジェクトフォルダを選択

プロジェクトがオープンされます。

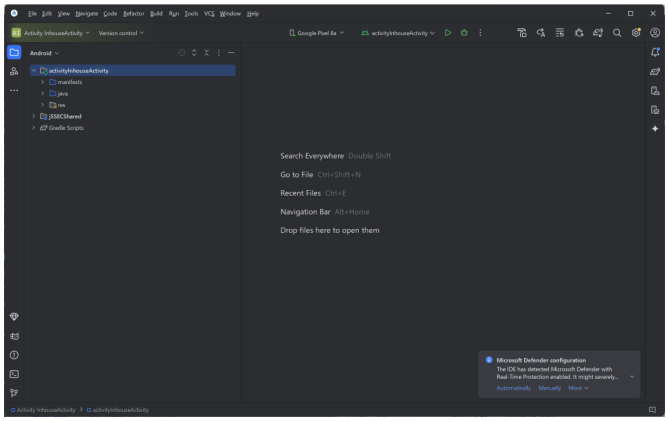

図 2.5.7　オープンの完了

サンプルコードから作成したアプリを Android 端末やエミュレーターで動作させるためには署名が必要です。この署名に使うデバッグ用の鍵ファイル”debug.keystore” を Android Studio のプロジェクトに設定します。

File -> Project Structure... をクリックします。

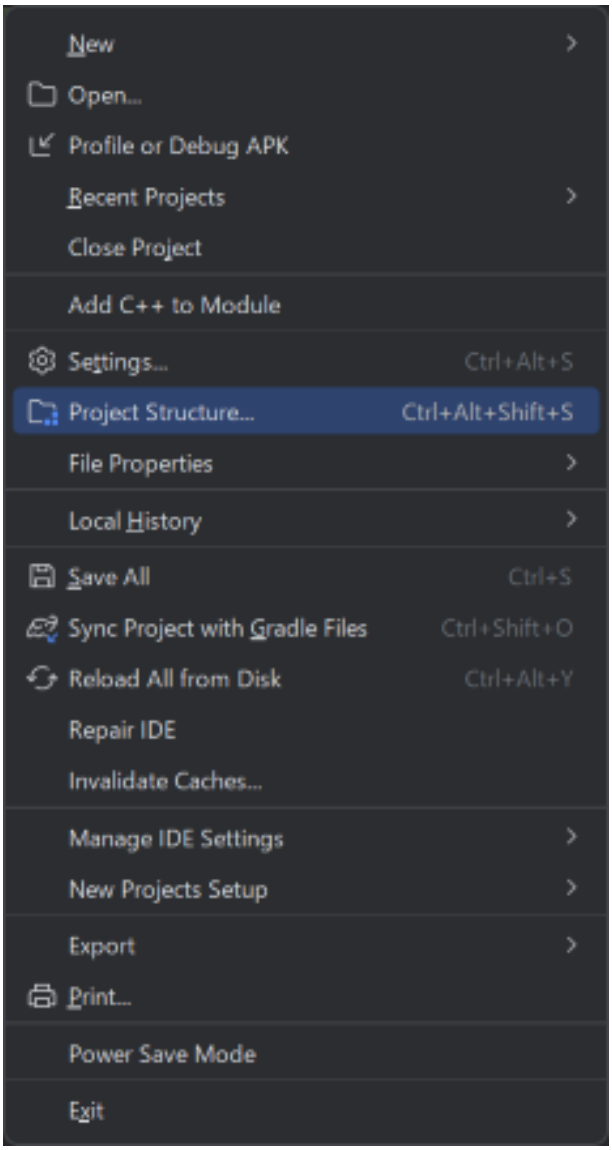

図 2.5.8　File -> Project Structure

Store File として”debug.keystore” を選択し OK をクリックします。

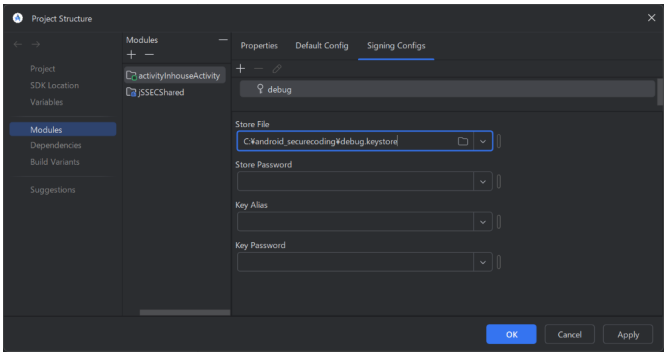

図 2.5.9　”debug.keystore” を選択

Build -> Make Project... をクリックします。

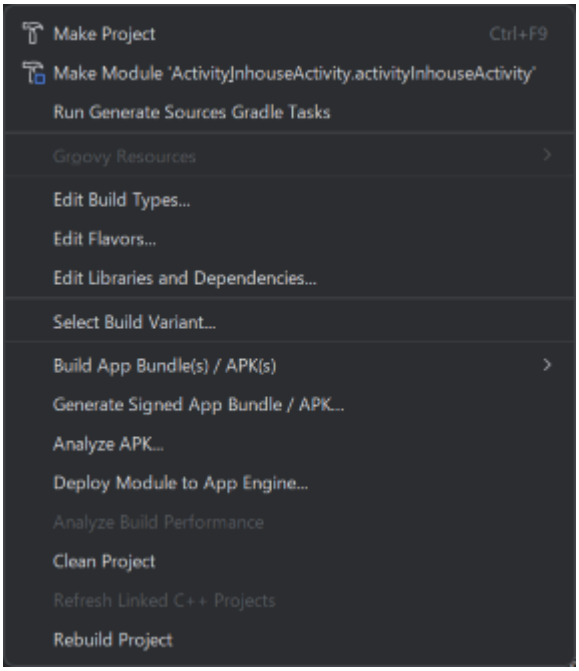

図 2.5.10　Make Project をクリック

3 セキュア設計・セキュアコーディングの基礎知識

このガイド文書は Android アプリ開発におけるセキュリティ Tips をまとめるものであるが、この章では Android スマートフォン／タブレットを例に一般的なセキュア設計・セキュアコーディングの基礎知識を扱う。後続の章において一般的なセキュア設計・セキュアコーディングの解説が必要なときに、本章の記事を参照するため、後続の章を読み進める前に本章の内容に一通り目を通しておくことをお勧めする。

3.1 Android アプリのセキュリティ

システムやアプリのセキュリティについて検討するとき、定番の考え方のフレームワークがある。まずそのシステムやアプリにおいて守るべき対象を把握する。これを「資産」と呼ぶ。次にその資産を脅かす攻撃を把握する。これを「脅威」と呼ぶ。最後に「資産」を「脅威」から守るための施策を検討・実施する。この施策を「対策」と呼ぶ。

ここで「対策」とは、システムやアプリに適切なセキュア設計・セキュアコーディングを施すことであり、このガイド文書では 4 章以降でこれを扱っている。本節では「資産」および「脅威」について焦点を当てる。

3.1.1 「資産」　守るべき対象

システムやアプリにおける「守るべき対象」には「情報」と「機能」の 2 つがある。これらをそれぞれ「情報資産」と「機能資産」と呼ぶ。「情報資産」とは、許可された人だけが参照や変更ができる情報のことであり、それ以外の人には一切参照や変更ができてはならない情報のことである。「機能資産」とは許可された人だけが利用できる機能のことであり、それ以外の人には一切利用できてはならない機能のことである。

以下、Android スマートフォン／タブレットにおける情報資産と機能資産にどのようなものがあるかを紹介する。Android アプリや Android スマートフォン／タブレットを活用したシステムを開発するときの資産の洗い出しの参考にしてほしい。以降では、Android スマートフォン／タブレットを総称して Android スマートフォンと呼ぶ。

3.1.1.1 Android スマートフォンにおける情報資産

表 3.1.1 および 表 3.1.2 は Android スマートフォンに入っている情報の一例である。これらの情報はスマートフォンユーザーに関する個人情報、プライバシー情報またはそれらに類する情報に該当するため適切な保護が必要である。

表 3.1.1　Android スマートフォンが管理する情報の例

情報	備考
電話番号	スマートフォン自身の電話番号
通話履歴	受発信の日時や相手番号
IMEI	スマートフォンの端末 ID
IMSI	回線契約者 ID
センサー情報	GPS、地磁気、加速度...
各種設定情報	WiFi 設定値...
アカウント情報	各種アカウント情報、認証情報...
メディアデータ	写真、動画、音楽、録音...
...	

表 3.1.2　アプリが管理する情報の例

情報	備考
電話帳	知人の連絡先
E メールアドレス	ユーザーのメールアドレス
Web ブックマーク	ブックマーク
Web 閲覧履歴	閲覧履歴
カレンダー	予定、ToDo、イベント...
Facebook	SNS コンテンツ...
X	SNS コンテンツ...
...	

表 3.1.1 の情報は主に Android スマートフォン本体または SD カードに含まれる情報であり、表 3.1.2 の情報は主にアプリが管理する情報である。特に 表 3.1.2 の情報については、アプリがインストールされればされるほど、どんどん本体の中に増えていくことになるのである。

表 3.1.3 は電話帳の 1 件のエントリに含まれる情報である。この情報はスマートフォンユーザーに関する情報ではなく、スマートフォンユーザーの知人、友人等に関する情報である。つまりスマートフォンにはその利用者であるユーザーのみならず、ほかの人々の情報も含まれていることに注意が必要だ。

表 3.1.3　電話帳（Contacts）の 1 件のエントリに含まれる情報の例

情報	内容
電話番号	自宅、携帯電話、仕事、FAX、MMS...
E メールアドレス	自宅、仕事、携帯電話...
プロフィール画像	サムネール画像、大きな画像...
インスタントメッセンジャー	メッセージ、WhatsApp、LINE、Facebook Messenger、WeChat、Telegram...
ニックネーム	略称、イニシャル、旧姓、別名...
住所	国、郵便番号、地域、地方、町、通り...
グループ	お気に入り、家族、友達、同僚...
ウェブサイト	ブログ、プロフィールサイト、ホームページ、FTP サーバー、自宅、会社...
イベント	誕生日、記念日、その他...
関係する人物	配偶者、子供、父、母、マネージャー、助手、 同棲関係、パートナー...
SIP アドレス	自宅、仕事、その他...
...	...

これまでの説明では主にスマートフォンユーザーの情報を紹介してきたが、アプリはユーザー以外の情報も扱っている。図 3.1.1 は 1 つのアプリが管理している情報を表しており、大きく分けるとプログラム部分とデータ部分に分かれる。プログラム部分は主にアプリメーカーの情報であり、データ部分は主にユーザーの情報である。アプリメーカーの情報の中には、勝手にユーザーに利用されたくない情報もあり得るため、そうした情報についてはユーザーが参照・変更できないような保護が必要である。

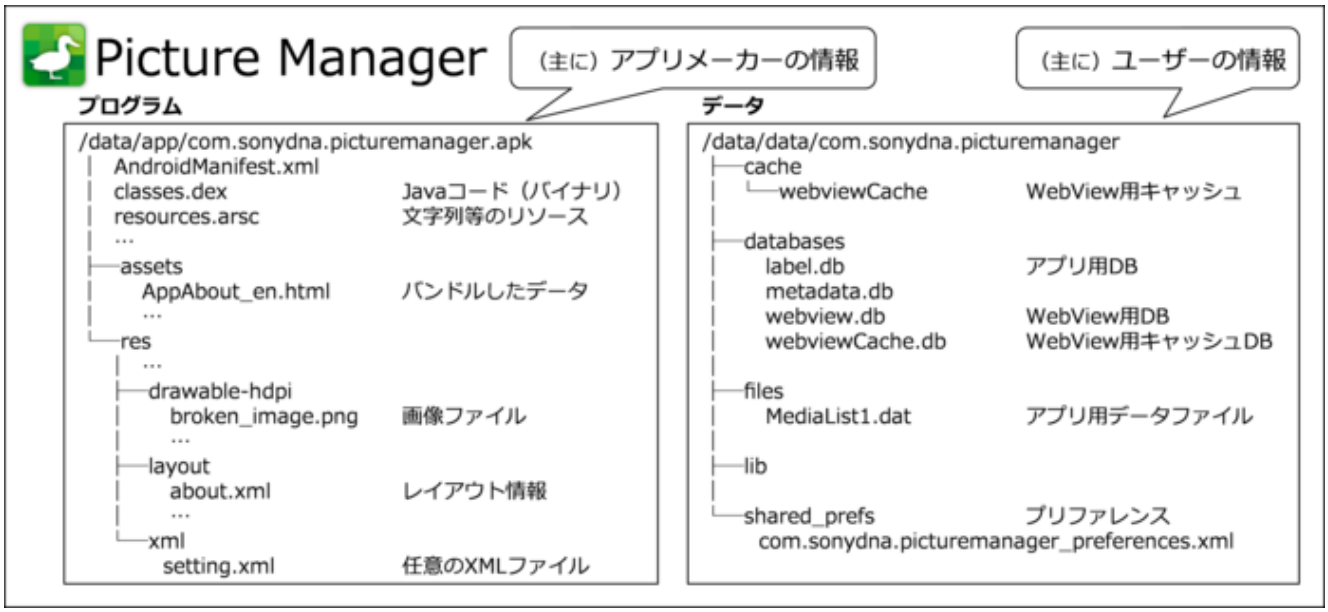

図 3.1.1　アプリが抱えている情報

Android アプリを作る場合には図 3.1.1 のようなアプリ自身が管理する情報のみならず、表 3.1.1 、表 3.1.2 、表 3.1.3 のような Android スマートフォン本体や他のアプリから取得した情報に関しても適切に保護する必要があることにも注意が必要だ。

3.1.1.2 Android スマートフォンにおける機能資産

表 3.1.4 は、Android OS がアプリに提供する機能の一例である。これらの機能がマルウェア等に勝手に利用されてしまうとユーザーの意図しない課金が生じたり、プライバシーが損なわれるなどの被害が生じたりする。そのため情報資産と同様に、こうした機能資産も適切に保護されなければならない。

表 3.1.4　Android OS がアプリに提供する主な機能例

機能	説明
SMS メッセージ送受信	SMS の送信や受信を行う
カメラ撮影	写真や動画を撮影する
電話発信	通話を開始する
音量変更	端末の音量を調整する
ネットワーク通信	インターネットやローカルネットワークに接続する
電話番号・端末情報取得	電話番号や端末の状態を取得する
位置情報取得（GPS 等）	現在地を取得する
SD カード書き込み	外部ストレージにデータを書き込む
Bluetooth 通信	Bluetooth 機器と通信する
システム設定変更	端末の各種設定を変更する
NFC/FeliCa 通信	近距離無線通信を利用する
ログデータ取得	システムやアプリのログを読み取る
インターネット通話（SIP）	SIP を利用した通話を行う
実行中アプリ情報取得	現在動作中のアプリ情報を取得する
その他	上記以外にも多様な機能が存在する

Android OS がアプリに提供する機能に加え、Android アプリのアプリ間連携機能も機能資産に含まれる。Android アプリはそのアプリ内で実現している機能を他のアプリから利用できるように提供することができ、このような仕組みをアプリ間連携と呼んでいる。この機能は便利である反面、Android アプリの開発者がセキュアコーディングの知識がないために、アプリ内部だけで利用する機能を誤って他のアプリから利用できるようにしてしまっているケースもある。他のアプリから利用できる機能の内容によっては、マルウェアから利用されては困ることもあるため、意図したアプリだけから利用できるように適切な保護が必要となることがある。

本ガイドにおけるサンプルコードでは、別のアプリに対してクエリまたは操作を行うサンプルについては、<queries> 要素でアクセス定義を行っている。これは Android 11 におけるパッケージアクセスの仕様 で導入された、最小限の権限の原則に従ったものである。

また、Google Play では、SMS や通話履歴に関する権限グループなど、危険性が高い、または機密情報に関わる権限の使用が制限されており、ポリシーの要件を満たしていない、もしくは申告フォームを提出していないアプリは、Google Play から削除される場合がある[*1]。

3.1.2 「脅威」　資産を脅かす攻撃

前節では Android スマートフォンにおける資産について解説した。ここではそれらの脅威、つまり資産を脅かす攻撃について解説する。資産が脅かされるとは簡単に言えば、情報資産が他人に勝手に参照・変更・削除・作成されることを言い、機能資産が他人に勝手に利用されることを言う、といった具合だ。こうした資産を直接的および間接的に操作する攻撃

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

行為を「脅威」と呼ぶ。また攻撃行為を行う人や物のことを「脅威源」と呼ぶ。攻撃者やマルウェアは脅威源であって脅威ではない。攻撃者やマルウェアが行う攻撃行為のことを脅威と呼ぶのである。これら用語間の関係を 図 3.1.2 に示す。

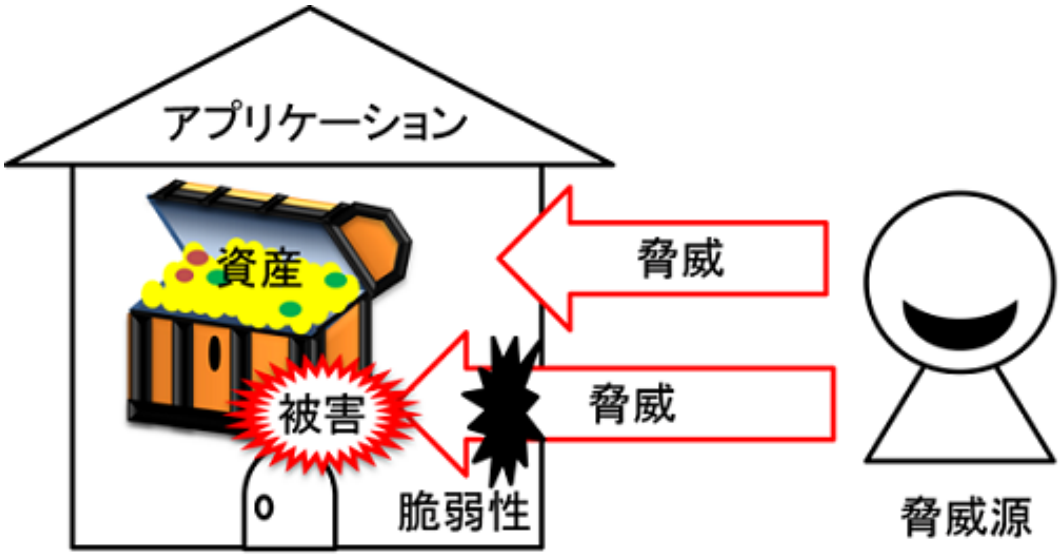

図 3.1.2　資産、脅威、脅威源、脆弱性、被害の関係

図 3.1.3 は Android アプリが動作する一般的な環境を表現したものだ。以降ではこの図をベースにして Android アプリにおける脅威の説明を展開するため、初めにこの図の見方を解説する。

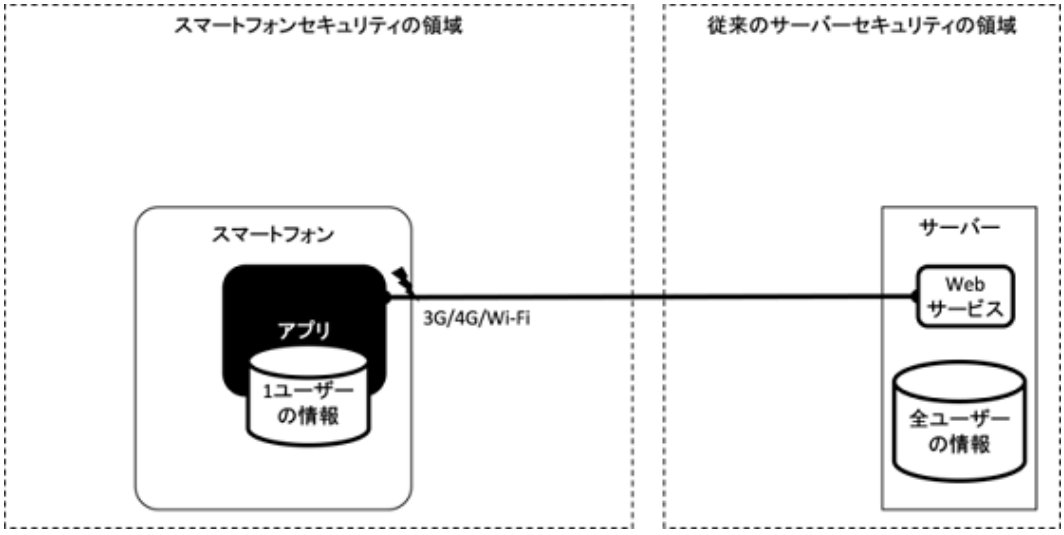

図 3.1.3　Android アプリが動作する一般的な環境

図の左右にスマートフォンとサーバーを配置している。スマートフォンやサーバーは 3G/4G/5G/Wi-Fi およびインターネットを経由して通信している。スマートフォンの中には複数のアプリが存在するが、以降の説明で 1 つのアプリに関する脅威を説明するため、この図では 1 つのアプリに絞って説明している。スマートフォン上のアプリはそのユーザーの情報を主に扱うが、サーバー上の Web サービスは全ユーザーの情報を集中管理することを表現している。そのため従来同様にサーバーセキュリティの重要性は変わらない。サーバーセキュリティについては、このガイド文書ではスコープ外であるため言及しない。

以降ではこの図を使って Android アプリにおける脅威を説明していく。

3.1.2.1 ネットワーク上の第三者による脅威

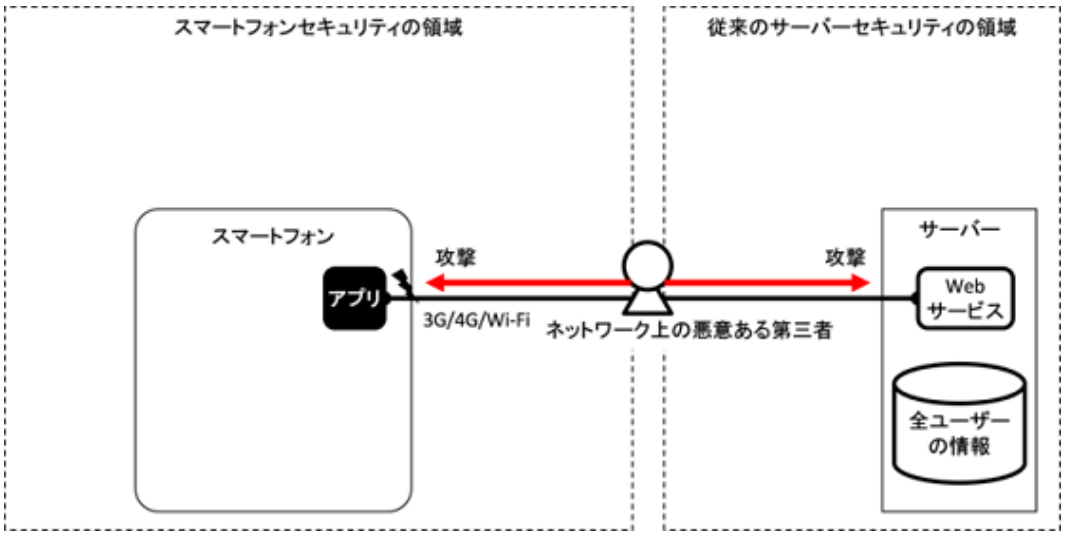

図 3.1.4　ネットワーク上の悪意ある第三者がアプリを攻撃する

スマートフォンアプリはユーザーの情報をサーバーで管理する形態が一般的である。そのため情報資産がネットワーク上を移動することになる。図 3.1.4 に示すように、ネットワーク上の悪意ある第三者は通信中の情報を参照（盗聴）したり、情報を変更（改ざん）したりしようとする。また本物のサーバーになりすまして、アプリの通信相手になろうとする。もちろん従来同様、ネットワーク上の悪意ある第三者はサーバーも攻撃する。

3.1.2.2 ユーザーがインストールしたマルウェアによる脅威

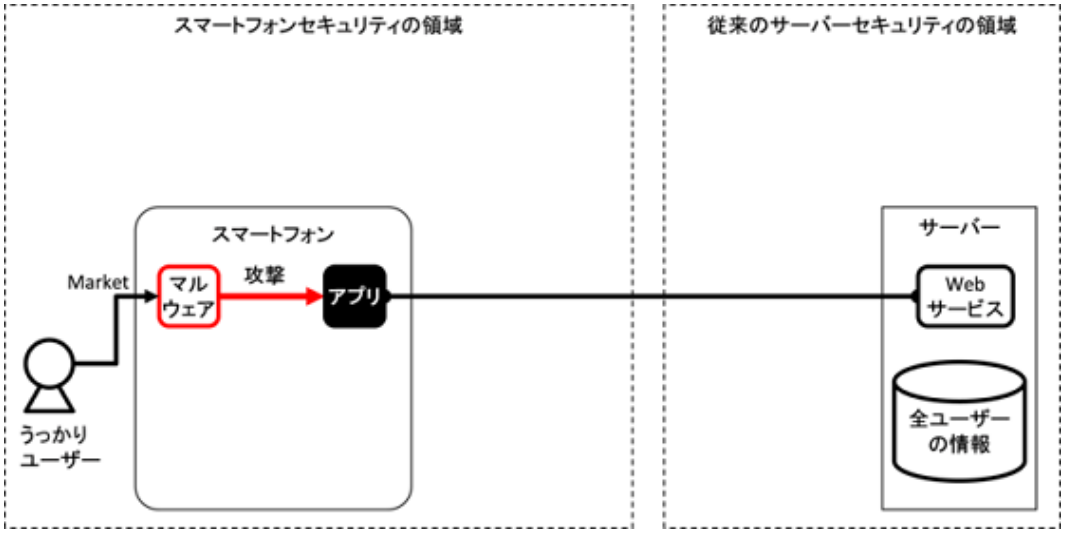

図 3.1.5　ユーザーがインストールしてしまったマルウェアがアプリを攻撃する

スマートフォンは多種多様なアプリをマーケットから入手しインストールすることで機能拡張できることがその最大の特徴である。ユーザーがうっかりマルウェアアプリをインストールしてしまうこともある。図 3.1.5 が示すように、マルウェアはアプリ間連携機能やアプリの脆弱性を悪用してアプリの情報資産や機能資産にアクセスしようとする。

3.1.2.3 アプリの脆弱性を悪用する攻撃ファイルによる脅威

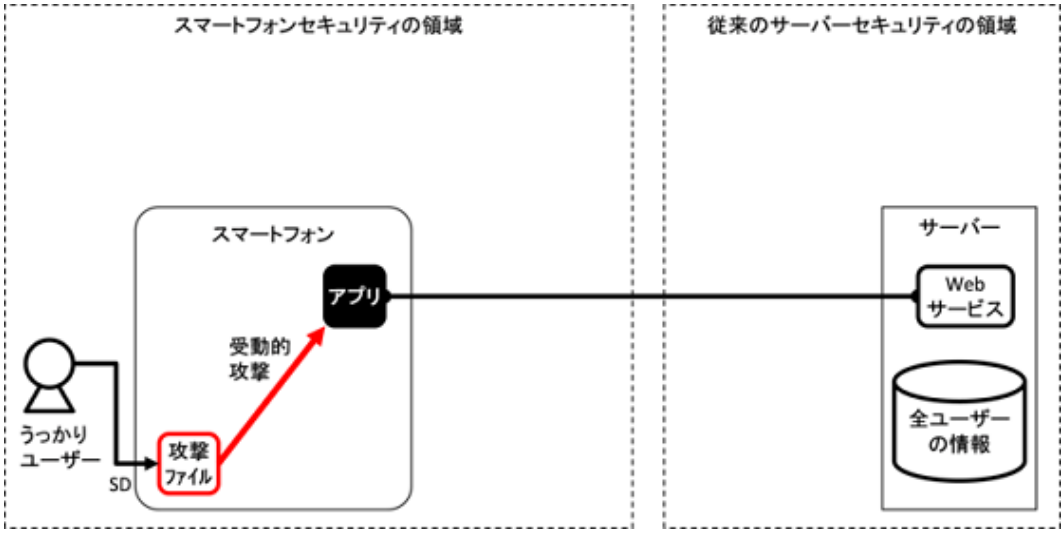

図 3.1.6　アプリの脆弱性を悪用する攻撃ファイルがアプリを攻撃する

インターネット上には音楽や写真、動画、文書など、様々なタイプのファイルが大量に公開されており、ユーザーがそれらのファイルを SD カードにダウンロードし、スマートフォンで利用する形態が一般的である。またスマートフォンで受信したメールに添付されるファイルを利用する形態も一般的である。これらのファイルは閲覧用や編集用のアプリでオープンされ利用される。

こうしたファイルを処理するアプリの機能に脆弱性があると、攻撃ファイルにより、そのアプリの情報資産や機能資産が悪用されてしまう。特に複雑なデータ構造を持ったファイル形式の処理においては脆弱性が入り込みやすい。攻撃ファイルは巧みに脆弱性を悪用してアプリを操作し、攻撃ファイルの作成者の目的を達成する。

図 3.1.6 に示すように、攻撃ファイルは脆弱なアプリによってオープンされるまでは何もせず、いったんオープンされるとアプリの脆弱性を悪用した攻撃を始める。攻撃者が能動的に行う攻撃行為と比較して、このような攻撃手法を受動的攻撃（Passive Attack）と呼ぶ。

3.1.2.4 悪意あるスマートフォンユーザーによる脅威

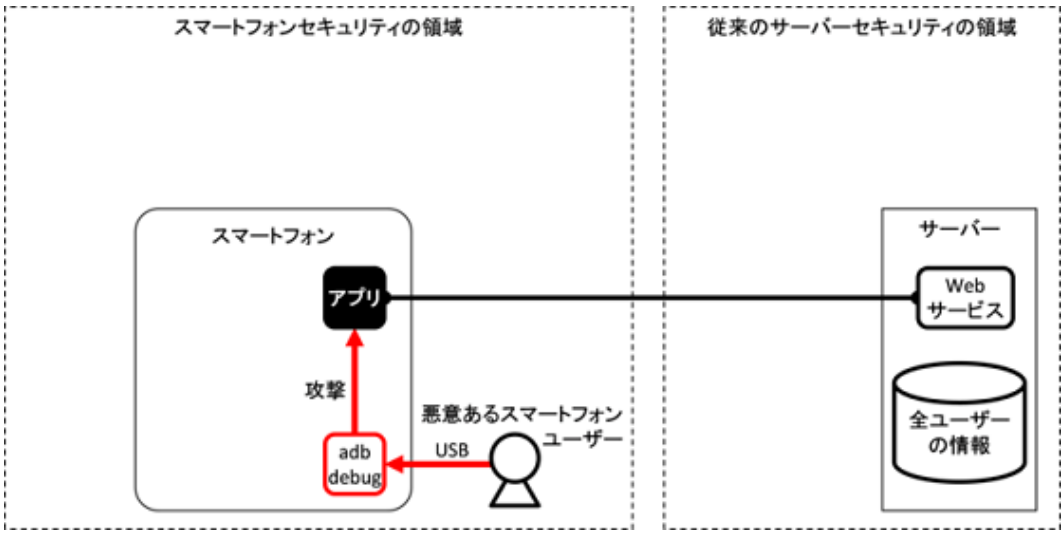

図 3.1.7　悪意あるスマートフォンユーザーがアプリを攻撃する

Android スマートフォンのアプリ開発においては、一般ユーザーに対してアプリを開発、解析する環境や機能が公式に提供されている。提供されている機能の中でも、特に ADB と呼ばれる充実したデバッグ機能は、誰でも何の登録・審査もなく利用可能であり、この機能により Android スマートフォンユーザーは OS 解析行為やアプリ解析行為を容易に行うことができる。

図 3.1.7 に示すように、悪意あるスマートフォンユーザーは ADB 等のデバッグ機能を利用してアプリを解析し、アプリが抱える情報資産や機能資産にアクセスしようとする。アプリが抱える資産がユーザー自身のものである場合には問題とならないが、アプリメーカー等ユーザー以外のステークホルダーの資産である場合に問題となる。このようにスマートフォンのユーザー自身が悪意を持ってアプリ内の資産を狙うことがあることにも注意が必要だ。

3.1.2.5 スマートフォンの近くにいる第三者による脅威

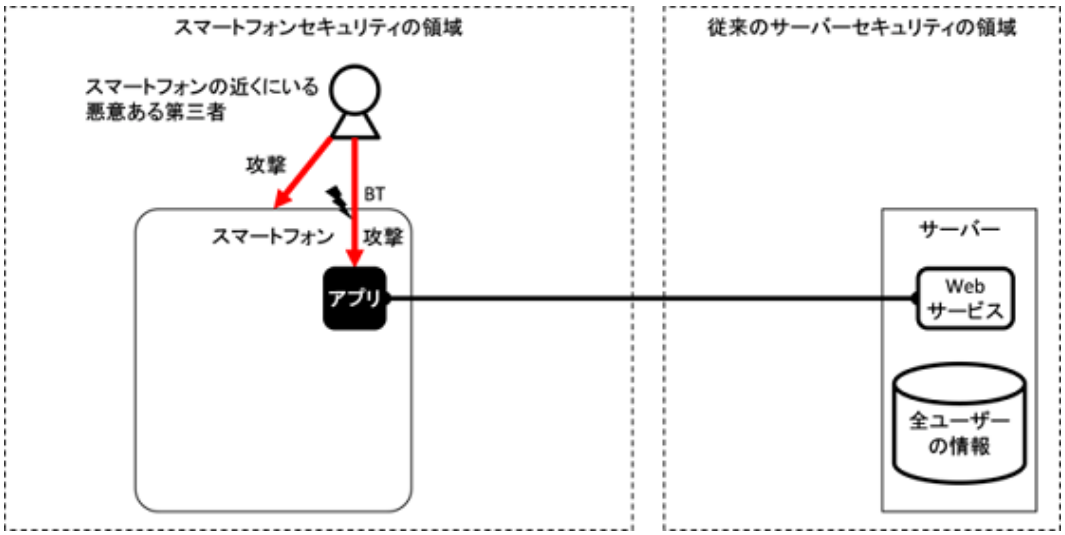

図 3.1.8　スマートフォンの近くにいる悪意ある第三者がアプリを攻撃する

スマートフォンはその携帯性の良さや、Bluetooth 等の近距離無線通信機能を標準搭載していることから、物理的にスマートフォンの近くにいる悪意ある第三者から攻撃され得ることも忘れてはならない。攻撃者はユーザーが入力中のパスワードを肩越しに覗き見したり、図 3.1.8 に示すように、Bluetooth 通信機能を持つアプリに対して Bluetooth でアクセスしたり、スマートフォン自体を盗んだり破壊したりする。特にスマートフォン自体の窃盗や破壊については、機密度の高さからスマートフォン内から一切外に出さない運用としている情報資産が失われてしまう脅威であり、アプリ設計の段階で見過ごされてしまうこともあるので注意が必要だ。

3.1.2.6 様々な脅威

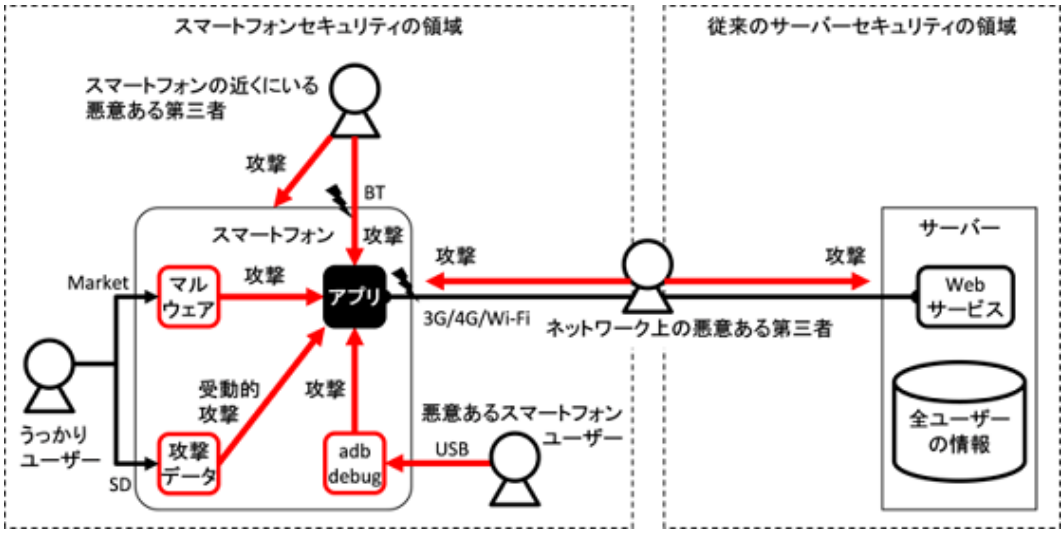

図 3.1.9　スマートフォンアプリは様々な攻撃にさらされている

図 3.1.9 はこれまで説明した脅威をひとまとめにした図である。このようにスマートフォンアプリを取り巻く脅威には様々なものがあり、この図はそのすべてを書き出しているものではない。日々の情報収集により、Android アプリを取り巻く脅威について認識を広め、アプリのセキュア設計・セキュアコーディングに活かしていく努力が必要である。一般社団法人日本スマートフォンセキュリティ協会が作成した次の文書もスマートフォンの脅威について役立つ情報を提供しているので参考にしていただきたい。

『スマートフォン＆タブレットの業務利用に関するセキュリティガイドライン』【第二版】

https://www.jssec.org/dl/guidelines_v2.pdf

https://www.jssec.org/dl/guidelines2012Enew_v1.0.pdf (English)

『スマートフォンネットワークセキュリティ実装ガイド』 【第一版】

https://www.jssec.org/dl/NetworkSecurityGuide1.pdf

『スマートフォンの業務利用におけるクラウド活用ガイド』 【ベータ版】

https://www.jssec.org/dl/cloudguide2012_beta.pdf

『MDM 導入・運用検討ガイド』【第一版】

https://www.jssec.org/dl/MDMGuideV1.pdf

3.1.3 資産分類と保護施策

前節までで解説した通り Android スマートフォンには様々な脅威が存在する。それらの脅威から、アプリが扱うすべての資産を保護することは、開発にかかる時間や技術的限界などから困難な場合がある。そのため、Android アプリ開発者は、アプリが扱う資産の重要度や容認される被害レベルを判断基準とした優先度に応じて、資産に対する妥当な対策を検討することが必要になる。

資産毎の保護施策を決めるため、アプリが扱うそれぞれの資産について、その重要度・保護に関する法的根拠・被害発生時の影響・開発者（または組織）の社会的責任などを検討した上で、資産を分類し、分類毎に保護施策のレベルを定める。これが、それぞれの資産をどのように扱い、どのような対策を施すかを決める判断基準となる。この分類はアプリ開発者（または組織）として資産をどのように扱い保護するかを定める基準そのものであるため、アプリ開発者（または組織）がそれぞれの事情に合わせて、分類方法や対策内容を定める必要がある。

参考までに、本ガイドにおける資産分類と保護施策のレベルを以下に示す。

表 3.1.5　資産分類と保護施策のレベル

資産分類	資産のレベル	保護施策のレベル
高位[2]	資産が被害にあった場合、組織または個人の活動に致命的または壊滅的な影響をあたえるもの。例）資産が被害にあった場合、当該組織が事業を継続できなくなるレベル。	Android OS のセキュリティモデルを破る、root 権限を奪取した状態からの攻撃や APK の dex 部分を改造するといった、高度な攻撃に対しても保護する。UX 等の他要素よりもセキュリティ確保を優先する。
中位	資産が被害にあった場合、組織または個人の活動に重大な影響をあたえるもの。例）資産が被害にあった場合、当該組織の利益が悪化し、事業に影響を及ぼすレベル。	Android OS のセキュリティモデルを活用し、その範囲内で保護する。UX 等の他要素よりもセキュリティ確保を優先する。
低位	資産が被害にあった場合、組織または個人の活動に限定的な影響をあたえるもの。例）資産が被害にあった場合、当該組織の利益に影響を与え得るが、他の要素により利益の補てんが可能なレベル。	UX 等の他要素とセキュリティを比較し、他要素を優先しても良い。

本ガイドにおける資産分類と保護施策は、基本的には root 権限が奪われていないセキュアな Android 端末を前提とし、Android OS のセキュリティモデルを活用したセキュリティ施策を基準にしている。具体的には、資産分類で中位レベル以下の資産に対して Android OS のセキュリティモデルが機能していることを前提に、Android OS のセキュリティモデルを活用した保護施策を想定している。

3.1.4 センシティブな情報

本ガイドのこれ以降の文章において情報資産を「センシティブな情報」と表現している。前節で述べたようにアプリが扱う個々の情報資産ごとに資産のレベルや保護施策のレベルを判断しなければならない。

[2] 高位レベルの資産は、root 権限が奪取された状態からの攻撃や、APK 解析・改造による攻撃といった、Android OS のセキュリティモデルが破られた状態での攻撃からも、保護が必要な資産であると想定している。このような資産を守るためには、Android OS のセキュリティモデルが活用できないため、暗号化、難読化、ハードウェア支援、サーバー支援など複数の手段を組み合わせて高度な防御設計をする。これはガイド文書に簡潔に書けるようなノウハウではないし、個別の状況に応じて適切な防御設計は異なるため本ガイドの対象外としている。root 権限奪取からの攻撃や APK 解析・改造などの高度な攻撃に対する保護が必要な場合は、Android の耐タンパ設計に詳しいセキュリティ専門家に相談することをお勧めする。

3.2 入力データの安全性を確認する

入力データの安全性確認はもっとも基礎的で効果の高いセキュアコーディング作法である。プログラムに入力されるデータのうち、攻撃者が直接的、間接的にそのデータの値を操作可能であるものはすべて、入力データの安全性確認が必要である。以下、Activity をプログラムに見立て、Intent を入力データに見立てた場合を例にして、入力データの安全性確認の在り方について解説する。

Activity が受け取った Intent には攻撃者が細工した　データが含まれている可能性がある。攻撃者はプログラマが想定していない形式・値のデータを送り付けることで、アプリの誤動作を誘発し、結果として何らかのセキュリティ被害を生じさせるのである。ユーザーも攻撃者の一人となり得ることも忘れてはならない。

Intent は action や data、extras などのデータで構成されるが、攻撃者が制御可能なデータはすべて気を付けなければならない。攻撃者が制御可能なデータを扱うコードでは、必ず次の事項を確認しなければならない。

(a) 受け取ったデータは、プログラマが想定した形式であって、値は想定の範囲内に収まっているか？

(b) 想定している形式、値のあらゆるデータを受け取っても、そのデータを扱うコードが想定外の動作をしないと保証できるか？

次の例は指定 URL の Internet 上の Web ページの HTML を取得し、画面上の TextView に表示するだけの簡単なサンプルである。しかしこれには不具合がある。

Internet 上の Web ページの HTML を TextView に表示するサンプルコード

```
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) { //...
```

(a) の観点で「urlstr が正しい URL である」ことを new URL() で MalformedURLException が発生しないことに　より確認している。しかしこれは不十分であり、urlstr に「file://～」形式の URL が指定されると Internet 上の Web ページではなく、内部ファイルシステム上のファイルを開いて TextView に表示してしまう。プログラマが想定した動作を保証していないため、(b) の観点を満たしていない。

次は改善例である。(a) の観点で「urlstr は正規の URL であって、protocol は http または https に限定される」ことを確認している。これにより (b) の観点でも url.openConnection().getInputStream() で Internet 経由の InputStream を取得することが保証される。

Internet 上の Web ページの HTML を TextView に表示するサンプルコードの修正版

```
TextView tv = (TextView) findViewById(R.id.textview);
InputStreamReader isr = null;
```

(次のページに続く)

(前のページからの続き)

```
char[] text = new char[1024];
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) { //...
```

入力データの安全性確認は Input Validation と呼ばれる基礎的なセキュアコーディング作法である。Input Validation という言葉の語感から (a) の観点のみ気を付けて (b) の観点を忘れてしまいがちである。データはプログラムに入ってきたときではなく、プログラムがそのデータを「使う」ときに被害が発生することに気を付けていただきたい。下記 URL もぜひ参考にしていただきたい。

IPA 「セキュア・プログラミング講座」

https://www.ipa.go.jp/security/awareness/vendor/programming/index.html

JPCERT CC 「Java セキュアコーディングスタンダード CERT/Oracle 版」

https://www.jpcert.or.jp/java-rules/

JPCERT CC 「Java Android アプリケーション開発へのルールの適用」

https://www.jpcert.or.jp/java-rules/android-j.html

4 安全にテクノロジーを活用する

Android で言えば Activity や SQLite など、テクノロジーごとにセキュリティ観点の癖というものがある。そうしたセキュリティの癖を知らずに設計、コーディングしていると思わぬ脆弱性をつくりこんでしまうことがある。この章では開発者が Android のテクノロジーを活用するシーンを想定した記事を扱う。

4.1 Activity を作る・利用する

4.1.1 サンプルコード

Activity がどのように利用されるかによって、Activity が抱えるリスクや適切な防御手段が異なる。ここでは、Activity がどのように利用されるかという観点で、Activity を 4 つのタイプに分類した。次の判定フローによって作成する Activity がどのタイプであるかを判断できる。なお、どのような相手を利用するかによって適切な防御手段が決まるため、Activity の利用側の実装についても合わせて説明する。

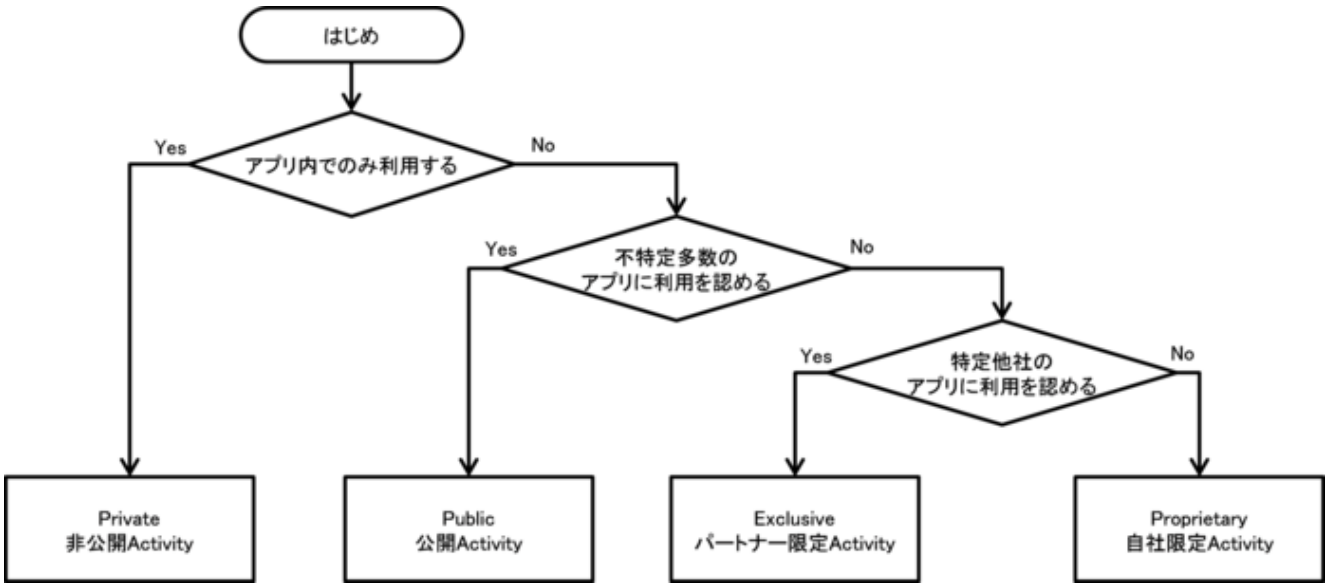

図 4.1.1　Activity タイプ選択フロー

4.1.1.1 非公開 Activity を作る・利用する

非公開 Activity は、同一アプリ内でのみ利用される Activity であり、もっとも安全性の高い Activity である。

同一アプリ内だけで利用される Activity（非公開 Activity）を利用する際は、クラスを指定する明示的 Intent を使えば誤って外部アプリに Intent を送信してしまうことがない。ただし、Activity を呼び出す際に使用する Intent は第三者によって読み取られる恐れがある。そのため、Activity に送信する Intent にセンシティブな情報を格納する場合には、その情報が悪意のある第三者に読み取られることのないように、適切な対応を実施する必要がある。

以下に非公開 Activity を作る側のサンプルコードを示す。

ポイント (Activity を作る)：

1. taskAffinity を指定しない
2. launchMode を指定しない
3. exported=”false” により、明示的に非公開設定する
4. 同一アプリからの Intent であっても、受信 Intent の安全性を確認する
5. 利用元アプリは同一アプリであるから、センシティブな情報を返送してよい

Activity を非公開設定するには、AndroidManifest.xml の activity 要素の exported 属性を false と指定する。

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" >

    <!-- 非公開 Activity -->
    <!-- ★ポイント 1★ taskAffinity を指定しない -->
    <!-- ★ポイント 2★ launchMode を指定しない -->
    <!-- ★ポイント 3★ exported="false"により、明示的に非公開設定する -->
    <activity
        android:name=".PrivateActivity"
        android:label="@string/app_name"
        android:exported="false" />

    <!-- ランチャーから起動する公開 Activity -->
    <activity
        android:name=".PrivateUserActivity"
        android:label="@string/app_name"
        android:exported="true" >
      <intent-filter>
        <action android:name="android.intent.action.MAIN" />
```

(次のページに続く)

(前のページからの続き)

```
        <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);

        // ★ポイント 4★ 同一アプリからの Intent であっても、受信 Intent の安全性を確認する
        // サンプルにつき割愛。「3.2 入力データの安全性を確認する」を参照。
        String param = getIntent().getStringExtra("PARAM");
        Toast.makeText(this, String.format("パラメータ「%s」を受け取った。", param),
                       Toast.LENGTH_LONG).show();
    }
```

(次のページに続く)

(前のページからの続き)

```
    public void onReturnResultClick(View view) {

        // ★ポイント 5★ 利用元アプリは同一アプリであるから、センシティブな情報を返送してよい
        Intent intent = new Intent();
        intent.putExtra("RESULT", "センシティブな情報");
        setResult(RESULT_OK, intent);
        finish();
    }
}
```

次に非公開 Activity を利用する側のサンプルコードを示す。

ポイント (Activity を利用する) :

6. Activity に送信する Intent には、フラグ FLAG_ACTIVITY_NEW_TASK を設定しない
7. 同一アプリ内 Activity はクラス指定の明示的 Intent で呼び出す
8. 利用先アプリは同一アプリであるから、センシティブな情報を putExtra() を使う場合に限り送信してもよい[*1]
9. 同一アプリ内 Activity からの結果情報であっても、受信データの安全性を確認する

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;
```

(次のページに続く)

[*1] ただし、ポイント 1, 2, 6 を遵守している場合を除いては Intent が第三者に読み取られるおそれがあることに注意する必要がある。詳細はルールブックセクションの 4.1.2.2.、4.1.2.3. を参照すること。

(前のページからの続き)

```
import android.view.View;
import android.widget.Toast;

public class PrivateUserActivity extends Activity {

    private static final int REQUEST_CODE = 1;

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

    public void onUseActivityClick(View view) {

        // ★ポイント 6★ Activity に送信する Intent には、フラグ FLAG_ACTIVITY_NEW_TASK を設定しない
        // ★ポイント 7★ 同一アプリ内 Activity はクラス指定の明示的 Intent で呼び出す
        Intent intent = new Intent(this, PrivateActivity.class);

        // ★ポイント 8★ 利用先アプリは同一アプリであるから、センシティブな情報を putExtra() を使う場合
に限り
        // 送信してもよい
        intent.putExtra("PARAM", "センシティブな情報");

        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");

            // ★ポイント 9★ 同一アプリ内 Activity からの結果情報であっても、受信データの安全性を確認す
る
            // サンプルにつき割愛。「3.2 入力データの安全性を確認する」を参照。
            Toast.makeText(this, String.format("結果「%s」を受け取った。", result),
                           Toast.LENGTH_LONG).show();
            break;
        }
    }
```

(次のページに続く)

(前のページからの続き)

```
}
```

4.1.1.2 公開 Activity を作る・利用する

公開 Activity は、不特定多数のアプリに利用されることを想定した Activity である。マルウェアが送信した Intent を受信することがあることに注意が必要である。また、公開 Activity を利用する場合には、送信する Intent がマルウェアに受信される、あるいは読み取られることがあることに注意が必要である。

以下に公開 Activity を作る側のサンプルコードを示す。

ポイント (Activity を作る) :

1. exported=”true” により、明示的に公開設定する
2. 受信 Intent の安全性を確認する
3. 結果を返す場合、センシティブな情報を含めない

```
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" >

    <!-- 公開 Activity -->
    <!-- ★ポイント 1★ exported="true"により、明示的に公開設定する -->
    <activity
        android:name=".PublicActivity"
        android:label="@string/app_name"
        android:exported="true" >

      <!-- Action 指定による暗黙的 Intent を受信するように Intent Filter を定義 -->
      <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
/*
```

(次のページに続く)

(前のページからの続き)

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

        // ★ポイント 2★ 受信 Intent の安全性を確認する
        // 公開 Activity であるため利用元アプリがマルウェアである可能性がある。
        // サンプルにつき割愛。「3.2 入力データの安全性を確認する」を参照。
        String param = getIntent().getStringExtra("PARAM");
        Toast.makeText(this,
                       String.format("パラメータ「%s」を受け取った。", param),
                       Toast.LENGTH_LONG).show();
    }

    public void onReturnResultClick(View view) {

        // ★ポイント 3★ 結果を返す場合、センシティブな情報を含めない
        // 公開 Activity であるため利用元アプリがマルウェアである可能性がある。
        // マルウェアに取得されても問題のない情報であれば結果として返してもよい。
```

(次のページに続く)

(前のページからの続き)

```
        Intent intent = new Intent();
        intent.putExtra("RESULT", "センシティブではない情報");
        setResult(RESULT_OK, intent);
        finish();
    }
}
```

次に公開 Activity を利用する側のサンプルコードを示す。

ポイント (Activity を利用する) :

4. センシティブな情報を送信してはならない
5. 結果を受け取る場合、結果データの安全性を確認する

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;
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
```

(次のページに続く)

(前のページからの続き)

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

    public void onUseActivityClick(View view) {

        try {
            // ★ポイント 4★ センシティブな情報を送信してはならない
            Intent intent = new Intent("org.jssec.android.activity.MY_ACTION");
            intent.putExtra("PARAM", "センシティブではない情報");
            startActivityForResult(intent, REQUEST_CODE);
        } catch (ActivityNotFoundException e) {
            Toast.makeText(this, "利用先 Activity が見つからない。", Toast.LENGTH_LONG).show();
        }
    }

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

        // ★ポイント 5★ 結果を受け取る場合、結果データの安全性を確認する
        // サンプルにつき割愛。「3.2 入力データの安全性を確認する」を参照。
        if (resultCode != RESULT_OK) return;
        switch (requestCode) {
        case REQUEST_CODE:
            String result = data.getStringExtra("RESULT");
            Toast.makeText(this,
                           String.format("結果「%s」を受け取った。", result),
                           Toast.LENGTH_LONG).show();
            break;
        }
    }
}
```

4.1.1.3 パートナー限定 Activity を作る・利用する

パートナー限定 Activity は、特定のアプリだけから利用できる Activity である。パートナー企業のアプリと自社アプリが連携してシステムを構成し、パートナーアプリとの間で扱う情報や機能を守るために利用される。

Activity を呼び出す際に使用する Intent は第三者によって読み取られる恐れがある。そのため、Activity に送信する Intent にセンシティブな情報を格納する場合には、その情報が悪意のある第三者に読み取られることのないように、適切な対応を実施する必要がある。

以下にパートナー限定 Activity を作る側のサンプルコードを示す。

ポイント (Activity を作る) :

1. taskAffinity を指定しない
2. launchMode を指定しない
3. Intent Filter を定義せず、exported=”true” を明示的に設定する
4. 利用元アプリの証明書がホワイトリストに登録されていることを確認する
5. パートナーアプリからの Intent であっても、受信 Intent の安全性を確認する
6. パートナーアプリに開示してよい情報に限り返送してよい

ホワイトリストを用いたアプリの確認方法については、「4.1.3.2. 利用元アプリを確認する」を参照すること。また、ホワイトリストに指定する利用先アプリの証明書ハッシュ値の確認方法は「5.2.1.3. アプリの証明書のハッシュ値を確認する方法」を参照すること。

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" >

    <!-- パートナー限定 Activity -->
    <!-- ★ポイント 1★ taskAffinity を指定しない -->
    <!-- ★ポイント 2★ launchMode を指定しない -->
    <!-- ★ポイント 3★ Intent Filter を定義せず、exported="true"を明示的に設定する -->
    <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
 *
```

(次のページに続く)

(前のページからの続き)

```
 * 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 {

    // ★ 4.1.1.3 - ポイント 4 ★ 利用元アプリの証明書がホワイトリストに登録されていることを確認する
    private static PkgCertWhitelists sWhitelists = null;
    private static void buildWhitelists(Context context) {
        boolean isdebug = Utils.isDebuggable(context);
        sWhitelists = new PkgCertWhitelists();

        // パートナーアプリ org.jssec.android.activity.partneruser の証明書ハッシュ値を登録
        sWhitelists.add("org.jssec.android.activity.partneruser", isdebug ?
            // debug.keystore の"androiddebugkey"の証明書ハッシュ値
            "0EFB7236 328348A9 89718BAD DF57F544 D5CCB4AE B9DB34BC 1E29DD26 F77C8255" :
            // keystore の"partner key"の証明書ハッシュ値
            "1F039BB5 7861C27A 3916C778 8E78CE00 690B3974 3EB8259F E2627B8D 4C0EC35A");

        // 以下同様に他のパートナーアプリを登録...
    }
    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);
```

(次のページに続く)

(前のページからの続き)

```
        // ★ 4.1.1.3 - ポイント 4★ 利用元アプリの証明書がホワイトリストに登録されていることを確認する
        if (!checkPartner(this, getCallingActivity().getPackageName())) {
            Toast.makeText(this, "利用元アプリはパートナーアプリではない。",
                           Toast.LENGTH_LONG).show();
            finish();
            return;
        }

        // ★ 4.1.1.3 - ポイント 5★ パートナーアプリからの Intent であっても、受信 Intent の安全性を確認する
        // サンプルにつき割愛。「3.2 入力データの安全性を確認する」を参照。
        Toast.makeText(this, "パートナーアプリからアクセスあり", Toast.LENGTH_LONG).show();
    }

    public void onReturnResultClick(View view) {

        // ★ 4.1.1.3 - ポイント 6★ パートナーアプリに開示してよい情報に限り返送してよい
        Intent intent = new Intent();
        intent.putExtra("RESULT", "パートナーアプリに開示してよい情報");
        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;
```

(次のページに続く)

(前のページからの続き)

```
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 は 32 バイト
        sha256 = sha256.toUpperCase();
        if (sha256.replaceAll("[0-9A-F]+", "").length() != 0)
            return false;    // 0-9A-F 以外の文字がある

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

    public boolean test(Context ctx, String pkgname) {
        // pkgname に対応する正解のハッシュ値を取得する
        String correctHash = mWhitelists.get(pkgname);

        // pkgname の実際のハッシュ値と正解のハッシュ値を比較する
        if (Build.VERSION.SDK_INT >= 28) {
            // ★ API Level >= 28 では Package Manager の API で直接検証が可能
            PackageManager pm = ctx.getPackageManager();
            return pm.hasSigningCertificate(pkgname,
                                            Utils.hex2Bytes(correctHash),
                                            CERT_INPUT_SHA256);
        } else {
            // API Level < 28 の場合は PkgCert の機能を利用する
            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);
            if (pkginfo.signatures.length != 1) return null;  // 複数署名は扱わない
            Signature sig = pkginfo.signatures[0];
            byte[] cert = sig.toByteArray();
```

(次のページに続く)

(前のページからの続き)

```
            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();
    }
}
```

次にパートナー限定 Activity を利用する側のサンプルコードを示す。ここではパートナー限定 Activity を呼び出す方法を説明する。

ポイント (Activity を利用する) :

7. 利用先パートナー限定 Activity アプリの証明書がホワイトリストに登録されていることを確認する
8. Activity に送信する Intent には、フラグ FLAG_ACTIVITY_NEW_TASK を設定しない
9. 利用先パートナー限定アプリに開示してよい情報は putExtra() を使う場合に限り送信してよい
10. 明示的 Intent によりパートナー限定 Activity を呼び出す
11. startActivityForResult() によりパートナー限定 Activity を呼び出す
12. パートナー限定アプリからの結果情報であっても、受信 Intent の安全性を確認する

ホワイトリストを用いたアプリの確認方法については、「4.1.3.2. 利用元アプリを確認する」を参照すること。また、ホワイトリストに指定する利用先アプリの証明書ハッシュ値の確認方法は「5.2.1.3. アプリの証明書のハッシュ値を確認する方法」を参照すること。

AndroidManifest.xml

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

(次のページに続く)

(前のページからの続き)

```
          >

  <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;
```

(次のページに続く)

(前のページからの続き)

```
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 {

    // ★ポイント 7★ 利用先パートナー限定 Activity アプリの証明書がホワイトリストに登録されていることを
確認する
    private static PkgCertWhitelists sWhitelists = null;
    private static void buildWhitelists(Context context) {
        boolean isdebug = Utils.isDebuggable(context);
        sWhitelists = new PkgCertWhitelists();

        // パートナー限定 Activity アプリ org.jssec.android.activity.partneractivity の証明書ハッ
シュ値を登録
        sWhitelists.add("org.jssec.android.activity.partneractivity", isdebug ?
            // debug.keystore の"androiddebugkey"の証明書ハッシュ値
            "0EFB7236 328348A9 89718BAD DF57F544 D5CCB4AE B9DB34BC 1E29DD26 F77C8255" :
            // keystore の"my company key"の証明書ハッシュ値
            "D397D343 A5CBC10F 4EDDEB7C A10062DE 5690984F 1FB9E88B D7B3A7C2 42E142CA");

        // 以下同様に他のパートナー限定 Activity アプリを登録...
    }
    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;

    // 利用先のパートナー限定 Activity に関する情報
    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);
```

(次のページに続く)

(前のページからの続き)

```
    }

    public void onUseActivityClick(View view) {

        // ★ポイント 7★ 利用先パートナー限定 Activity アプリの証明書がホワイトリストに登録されていること
を確認する
        if (!checkPartner(this, TARGET_PACKAGE)) {
            Toast.makeText(this,
                           "利用先 Activity アプリはホワイトリストに登録されていない。",
                            Toast.LENGTH_LONG).show();
            return;
        }

        try {
            Intent intent = new Intent();

            // ★ポイント 8★ Activity に送信する Intent には、フラグ FLAG_ACTIVITY_NEW_TASK を設定し
ない

            // ★ポイント 9★ 利用先パートナー限定アプリに開示してよい情報を putExtra() を使う場合に限
り送信してよい
            intent.putExtra("PARAM", "パートナーアプリに開示してよい情報");

            // ★ポイント 10★ 明示的 Intent によりパートナー限定 Activity を呼び出す
            intent.setClassName(TARGET_PACKAGE, TARGET_ACTIVITY);

            // ★ポイント 11★ startActivityForResult() によりパートナー限定 Activity を呼び出す
            startActivityForResult(intent, REQUEST_CODE);
        }
        catch (ActivityNotFoundException e) {
            Toast.makeText(this,
                           "利用先 Activity が見つからない。", 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");
```

(次のページに続く)

(前のページからの続き)

```
            // ★ポイント 12★ パートナー限定アプリからの結果情報であっても、受信 Intent の安全性を確認
する
            // サンプルにつき割愛。「3.2 入力データの安全性を確認する」を参照。
            Toast.makeText(this,
                    String.format("結果「%s」を受け取った。", 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;
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;
```

(次のページに続く)

(前のページからの続き)

```
        sha256 = sha256.replaceAll(" ", "");
        if (sha256.length() != 64)
            return false;    // SHA-256 は 32 バイト
        sha256 = sha256.toUpperCase();
        if (sha256.replaceAll("[0-9A-F]+", "").length() != 0)
            return false;    // 0-9A-F 以外の文字がある

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

    public boolean test(Context ctx, String pkgname) {
        // pkgname に対応する正解のハッシュ値を取得する
        String correctHash = mWhitelists.get(pkgname);

        // pkgname の実際のハッシュ値と正解のハッシュ値を比較する
        if (Build.VERSION.SDK_INT >= 28) {
            // ★ API Level >= 28 では Package Manager の API で直接検証が可能
            PackageManager pm = ctx.getPackageManager();
            return pm.hasSigningCertificate(pkgname,
                                            Utils.hex2Bytes(correctHash),
                                            CERT_INPUT_SHA256);
        } else {
            // API Level < 28 の場合は PkgCert の機能を利用する
            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
```

(次のページに続く)

(前のページからの続き)

```
 * 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);
            if (pkginfo.signatures.length != 1) return null;  // 複数署名は扱わない
            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;
        }
    }
```

(次のページに続く)

(前のページからの続き)

```
    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();
    }
}
```

4.1.1.4 自社限定 Activity を作る・利用する

自社限定 Activity は、自社以外のアプリから利用されることを禁止する Activity である。複数の自社製アプリでシステムを構成し、自社アプリが扱う情報や機能を守るために利用される。

Activity を呼び出す際に使用する Intent は第三者によって読み取られる恐れがある。そのため、Activity に送信する Intent にセンシティブな情報を格納する場合には、その情報が悪意のある第三者に読み取られることのないように、適切な対応を実施する必要がある。

以下に自社限定 Activity を作る側のサンプルコードを示す。

ポイント (Activity を作る):

1. 独自定義 Signature Permission を定義する
2. taskAffinity を指定しない
3. launchMode を指定しない
4. 独自定義 Signature Permission を要求宣言する
5. Intent Filter を定義せず、exported=”true” を明示的に設定する
6. 独自定義 Signature Permission が自社アプリにより定義されていることを確認する
7. 自社アプリからの Intent であっても、受信 Intent の安全性を確認する
8. 利用元アプリは自社アプリであるから、センシティブな情報を返送してよい
9. APK を Export するときに、利用元アプリと同じ開発者鍵で APK を署名する

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

  <!-- ★ポイント 1★ 独自定義 Signature Permission を定義する -->
  <permission
      android:name="org.jssec.android.activity.inhouseactivity.MY_PERMISSION"
```

(次のページに続く)

(前のページからの続き)

```
      android:protectionLevel="signature" />

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

    <!-- 自社限定 Activity -->
    <!-- ★ポイント 2★ taskAffinity を指定しない -->
    <!-- ★ポイント 3★ launchMode を指定しない -->
    <!-- ★ポイント 4★ 独自定義 Signature Permission を要求宣言する -->
    <!-- ★ポイント 5★ Intent Filter を定義せず、exported="true"を明示的に設定する -->
    <activity
        android:name=".InhouseActivity"
        android:exported="true"
        android:permission="org.jssec.android.activity.inhouseactivity.MY_PERMISSION" >
    </activity>
  </application>

</manifest>
```

InhouseActivity.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;
```

(次のページに続く)

(前のページからの続き)

```
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 {

    // 自社の Signature Permission
    private static final String MY_PERMISSION =
        "org.jssec.android.activity.inhouseactivity.MY_PERMISSION";

    // 自社の証明書のハッシュ値
    private static String sMyCertHash = null;
    private static String myCertHash(Context context) {
        if (sMyCertHash == null) {
            if (Utils.isDebuggable(context)) {
                // debug.keystore の"androiddebugkey"の証明書ハッシュ値
                sMyCertHash = "0EFB7236 328348A9 89718BAD DF57F544 D5CCB4AE B9DB34BC␣
↪1E29DD26 F77C8255";
            } else {
                // keystore の"my company key"の証明書ハッシュ値
                sMyCertHash = "D397D343 A5CBC10F 4EDDEB7C A10062DE 5690984F 1FB9E88B␣
↪D7B3A7C2 42E142CA";
            }
        }
        return sMyCertHash;
    }

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

        // ★ポイント 6★ 独自定義 Signature Permission が自社アプリにより定義されていることを確認する
        if (!SigPerm.test(this, MY_PERMISSION, myCertHash(this))) {
            Toast.makeText(this, "独自定義 Signature Permission が自社アプリにより定義されていな
い。",
                           Toast.LENGTH_LONG).show();
            finish();
            return;
        }

        // ★ポイント 7★ 自社アプリからの Intent であっても、受信 Intent の安全性を確認する
        // サンプルにつき割愛。「3.2 入力データの安全性を確認する」を参照
```

(次のページに続く)

(前のページからの続き)

```
        String param = getIntent().getStringExtra("PARAM");
        Toast.makeText(this, String.format("パラメータ「%s」を受け取った。", param),
                       Toast.LENGTH_LONG).show();
    }

    public void onReturnResultClick(View view) {

        // ★ポイント 8★ 利用元アプリは自社アプリであるから、センシティブな情報を返送してよい
        Intent intent = new Intent();
        intent.putExtra("RESULT", "センシティブな情報");
        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;
import android.os.Build;

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

public class SigPerm {

```

(次のページに続く)

(前のページからの続き)

```
    public static boolean test(Context ctx, String sigPermName, String correctHash) {
        if (correctHash == null) return false;
        correctHash = correctHash.replaceAll(" ", "");
        try{
            // sigPermName を定義したアプリのパッケージ名を取得する
            PackageManager pm = ctx.getPackageManager();
            PermissionInfo pi =
                pm.getPermissionInfo(sigPermName, PackageManager.GET_META_DATA);
            String pkgname = pi.packageName;
            // 非 Signature Permission の場合は失敗扱い
            if (pi.protectionLevel != PermissionInfo.PROTECTION_SIGNATURE) return false;
            // pkgname の実際のハッシュ値と正解のハッシュ値を比較する
            if (Build.VERSION.SDK_INT >= 28) {
                // ★ API Level >= 28 では Package Manager の API で直接検証が可能
                return pm.hasSigningCertificate(pkgname,
                                                Utils.hex2Bytes(correctHash),
                                                CERT_INPUT_SHA256);
            } else {
                // API Level < 28 の場合は PkgCert を利用し、ハッシュ値を取得して比較する
                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.
 */
```

(次のページに続く)

(前のページからの続き)

```java
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);
            if (pkginfo.signatures.length != 1) return null;  // 複数署名は扱わない
            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;
```

(次のページに続く)

(前のページからの続き)

```
        final StringBuilder hexadecimal = new StringBuilder();
        for (final byte b : data) {
            hexadecimal.append(String.format("%02X", b));
        }
        return hexadecimal.toString();
    }
}
```

★ポイント 9 ★ APK を Export するときに、利用元アプリと同じ開発者鍵で APK を署名する。

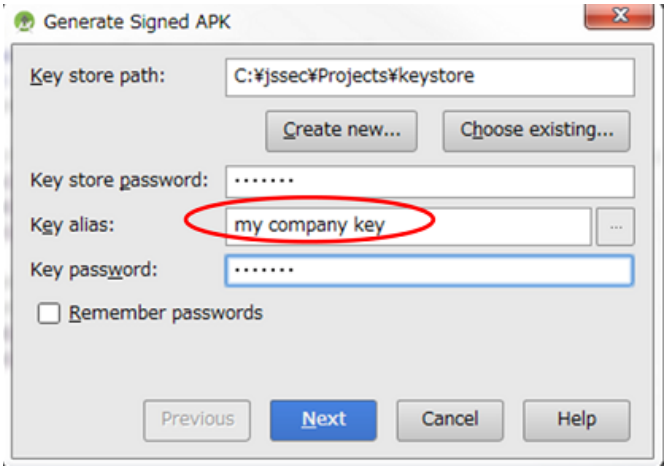

図 4.1.2　利用元アプリと同じ開発者鍵で APK を署名する

次に自社限定 Activity を利用する側のサンプルコードを示す。ここでは自社限定 Activity を呼び出す方法を説明する。

ポイント (Activity を利用する) :

10. 独自定義 Signature Permission を利用宣言する
11. 独自定義 Signature Permission が自社アプリにより定義されていることを確認する
12. 利用先アプリの証明書が自社の証明書であることを確認する
13. 利用先アプリは自社アプリであるから、センシティブな情報を putExtra() を使う場合に限り送信してもよい
14. 明示的 Intent により自社限定 Activity を呼び出す
15. 自社アプリからの結果情報であっても、受信 Intent の安全性を確認する
16. APK を Export するときに、利用先アプリと同じ開発者鍵で APK を署名する

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" />
```

(次のページに続く)

(前のページからの続き)

```
  </queries>

  <!-- ★ポイント 10★ 独自定義 Signature Permission を利用宣言する -->
  <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>
  </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;
```

(次のページに続く)

(前のページからの続き)

```
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 {

    // 利用先の Activity 情報
    private static final String TARGET_PACKAGE =
        "org.jssec.android.activity.inhouseactivity";
    private static final String TARGET_ACTIVITY =
        "org.jssec.android.activity.inhouseactivity.InhouseActivity";

    // 自社の Signature Permission
    private static final String MY_PERMISSION =
        "org.jssec.android.activity.inhouseactivity.MY_PERMISSION";

    // 自社の証明書のハッシュ値
    private static String sMyCertHash = null;
    private static String myCertHash(Context context) {
        if (sMyCertHash == null) {
            if (Utils.isDebuggable(context)) {
                // debug.keystore の"androiddebugkey"の証明書ハッシュ値
                sMyCertHash = "0EFB7236 328348A9 89718BAD DF57F544 D5CCB4AE B9DB34BC␣
↪1E29DD26 F77C8255";
            } else {
                // keystore の"my company key"の証明書ハッシュ値
                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);
```

(次のページに続く)

(前のページからの続き)

```
    }

    public void onUseActivityClick(View view) {

        // ★ポイント 11 ★ 独自定義 Signature Permission が自社アプリにより定義されていることを確認す
る
        if (!SigPerm.test(this, MY_PERMISSION, myCertHash(this))) {
            Toast.makeText(this, "独自定義 Signature Permission が自社アプリにより定義されていな
い。",
                           Toast.LENGTH_LONG).show();
            return;
        }

        // ★ポイント 12 ★ 利用先アプリの証明書が自社の証明書であることを確認する
        if (!PkgCert.test(this, TARGET_PACKAGE, myCertHash(this))) {
            Toast.makeText(this, "利用先アプリは自社アプリではない。", Toast.LENGTH_LONG).show();
            return;
        }

        try {
            Intent intent = new Intent();

            // ★ポイント 13 ★ 利用先アプリは自社アプリであるから、センシティブな情報を putExtra() を使
う場合に限り送信してもよい
            intent.putExtra("PARAM", "センシティブな情報");

            // ★ポイント 14 ★ 明示的 Intent により自社限定 Activity を呼び出す
            intent.setClassName(TARGET_PACKAGE, TARGET_ACTIVITY);
            startActivityForResult(intent, REQUEST_CODE);
        }
        catch (ActivityNotFoundException e) {
            Toast.makeText(this, "利用先 Activity が見つからない。", 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");
```

(次のページに続く)

(前のページからの続き)

```
            // ★ポイント 15 ★ 自社アプリからの結果情報であっても、受信 Intent の安全性を確認する
            // サンプルにつき割愛。「3.2 入力データの安全性を確認する」を参照。
            Toast.makeText(this, String.format("結果「%s」を受け取った。", 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{
            // sigPermName を定義したアプリのパッケージ名を取得する
            PackageManager pm = ctx.getPackageManager();
```

(次のページに続く)

(前のページからの続き)

```
            PermissionInfo pi =
                pm.getPermissionInfo(sigPermName, PackageManager.GET_META_DATA);
            String pkgname = pi.packageName;
            // 非 Signature Permission の場合は失敗扱い
            if (pi.protectionLevel != PermissionInfo.PROTECTION_SIGNATURE) return false;
            // pkgname の実際のハッシュ値と正解のハッシュ値を比較する
            if (Build.VERSION.SDK_INT >= 28) {
                // ★ API Level >= 28 では Package Manager の API で直接検証が可能
                return pm.hasSigningCertificate(pkgname,
                                                Utils.hex2Bytes(correctHash),
                                                CERT_INPUT_SHA256);
            } else {
                // API Level < 28 の場合は PkgCert を利用し、ハッシュ値を取得して比較する
                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;
```

(次のページに続く)

(前のページからの続き)

```
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);
            if (pkginfo.signatures.length != 1) return null;  // 複数署名は扱わない
            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();
    }
```

(次のページに続く)

(前のページからの続き)

```
}
```

★ポイント 16 ★ APK を Export するときに、利用先アプリと同じ開発者鍵で APK を署名する。

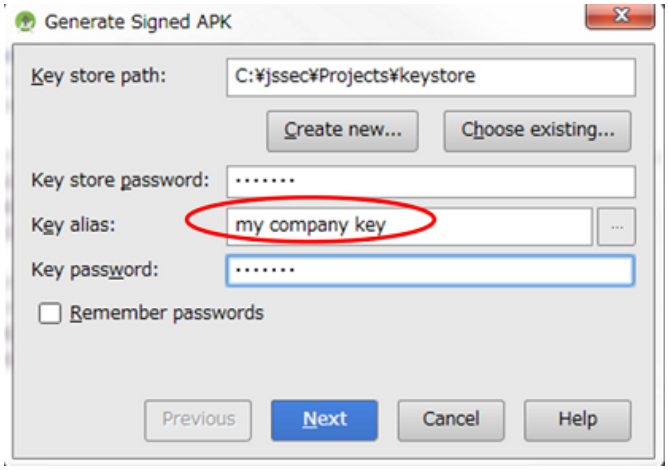

図 4.1.3　利用先アプリと同じ開発者鍵で APK を署名する

4.1.2 ルールブック

Activity を作る際、または Activity に Intent を送信する際には以下のルールを守ること。

1. アプリ内でのみ使用する *Activity* は非公開設定する（必須）
2. *taskAffinity* を指定しない（必須）
3. *launchMode* を指定しない（必須）
4. *Activity* に送信する *Intent* には *FLAG_ACTIVITY_NEW_TASK* を設定しない（必須）
5. 受信 *Intent* の安全性を確認する（必須）
6. 独自定義 *Signature Permission* は、自社アプリが定義したことを確認して利用する（必須）
7. 結果情報を返す場合には、返送先アプリからの結果情報漏洩に注意する（必須）
8. 利用先 *Activity* が固定できる場合は明示的 *Intent* で *Activity* を利用する（必須）
9. 利用先 *Activity* からの戻り *Intent* の安全性を確認する（必須）
10. 他社の特定アプリと連携する場合は利用先 *Activity* を確認する（必須）
11. 資産を二次的に提供する場合には、その資産の従来の保護水準を維持する（必須）
12. センシティブな情報はできる限り送らない（推奨）

4.1.2.1 アプリ内でのみ使用する **Activity** は非公開設定する（必須）

同一アプリ内からのみ利用される Activity は他のアプリから Intent を受け取る必要がない。またこのような Activity では開発者も Activity を攻撃する Intent を想定しないことが多い。このような Activity は明示的に非公開設定し、非公開 Activity とする。

```
AndroidManifest.xml
    <!-- 非公開 Activity -->
    <!-- ★ 4.1.1.1 - ポイント 3★ exported="false"により、明示的に非公開設定する -->
    <activity
        android:name=".PrivateActivity"
        android:label="@string/app_name"
        android:exported="false" />
```

同一アプリ内からのみ利用される Activity では Intent Filter を設置するような設計はしてはならない。Intent Filter の性質上、同一アプリ内の非公開 Activity を呼び出すつもりでも、Intent Filter 経由で呼び出したときに意図せず他アプリの Activity を呼び出してしまう可能性もある。詳細は、アドバンスト「4.1.3.1. *exported* 設定と *intent-filter* 設定の組み合わせ *(Activity* の場合*)*」を参照すること。

```
AndroidManifest.xml(非推奨)
    <!-- 非公開 Activity -->
    <!-- ★ 4.1.1.1 - ポイント 3★ exported="false"により、明示的に非公開設定する -->
    <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 **taskAffinity** を指定しない（必須）

Android では、Activity はタスクによって管理される。タスクの名前は、ルート Activity の持つアフィニティによって決定される。一方でルート以外の Activity に関しては、所属するタスクがアフィニティだけでは決定されず、Activity の起動モードにも依存する。詳細は「4.1.3.3. ルート *Activity* について」を参照すること。

デフォルト設定では各 Activity はパッケージ名をアフィニティとして持つ。その結果、タスクはアプリごとに割り当てられるので、同一アプリ内の全ての Activity は同一タスクに所属する。タスクの割り当てを変更するには、AndroidManifest.xml への明示的なアフィニティ記述や、Activity に送信する Intent へのフラグ設定をすればよい。ただし、タスクの割り当てを変更した場合は、異なるタスクに属する Activity に送信した Intent を別アプリによって読み出せる可能性がある。

センシティブな情報を含む送信 Intent および受信 Intent の内容を読み取られないようにするには、AndroidManifest.xml 内の application 要素および activity 要素で android:taskAffinity を指定せず、デフォルト (パッケージ名と同一) のままにすべきである。

以下に非公開 Activity の作成側と利用側における AndroidManifest.xml を示す。

```
AndroidManifest.xml
    <!-- ★ 4.1.1.1 - ポイント 1 ★ taskAffinity を指定しない -->
    <application
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name" >

        <!-- ★ 4.1.1.1 - ポイント 1 ★ taskAffinity を指定しない -->
        <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>

        <!-- 非公開 Activity -->
        <!-- ★ 4.1.1.1 - ポイント 1 ★ taskAffinity を指定しない -->
        <activity
            android:name=".PrivateActivity"
            android:label="@string/app_name"
            android:exported="false" />
    </application>
```

タスクとアフィニティの詳細な解説は、「Google Android プログラミング入門」[*2]、あるいは、Google Developers API Guide ”Tasks and Back Stack”[*3] の解説および「4.1.3.3. ルート *Activity* について」を参照すること。

4.1.2.3 launchMode を指定しない（必須）

Activity の起動モードとは、Activity を呼び出す際に、Activity のインスタンスの新規生成や、タスクの新規生成を制御するための設定である。デフォルト設定は”standard” である。”standard” 設定では、Intent を使って Activity を呼び出すときには常に新規インスタンスを生成し、タスクは呼び出し側 Activity が属するタスクに従い、新規にタスクが生成されることはない。タスクが新規に生成されると、呼び出しに使った Intent が別のアプリから読み取り可能になる。そのため、センシティブな情報を Intent に含む場合には、Activity の起動モードには”standard” を用いるべきである。

Activity の起動モードは AndroidManifest.xml 内にて android:launchMode で明示的に設定可能であるが、上記の理由により、各 Activity に対して android:launchMode を指定せず、値をデフォルトのまま”standard” とするべきである。

```
AndroidManifest.xml
        <!-- 非公開 Activity -->
        <!-- ★ 4.1.1.1 - ポイント 2 ★ launchMode を指定しない -->
        <!-- Activity には launchMode を指定せず、値をデフォルトのまま” standard” とする -->
        <activity
            android:name=".PrivateUserActivity"
```

(次のページに続く)

[*2] 江川、藤井、麻野、藤田、山田、山岡、佐野、竹端著「Google Android プログラミング入門」(アスキー・メディアワークス、2009 年 7 月)
[*3] https://developer.android.com/guide/components/tasks-and-back-stack.html

（前のページからの続き）

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

        <!-- 非公開 Activity -->
        <!-- ★ 4.1.1.1 - ポイント 2★ launchMode を指定しない -->
        <!-- Activity には launchMode を指定せず、値をデフォルトのまま”standard”とする -->
        <activity
            android:name=".PrivateActivity"
            android:label="@string/app_name"
            android:exported="false" />
    </application>
```

「4.1.3.3. ルート *Activity* について」を参照すること。

4.1.2.4 Activity に送信する Intent には FLAG_ACTIVITY_NEW_TASK を設定しない（必須）

Activity の起動モードは startActivity() あるいは startActivityForResult() の実行時にも変更することが可能であり、タスクが新規に生成される場合がある。そのため、Activity の起動モードを実行時に変更しないようにする必要がある。

Activity の起動モードを変更するには、setFlags() や addFlags() を用いて Intent にフラグを設定し、その Intent を startActivity() または startActivityForResult() の引数とする。タスクを新規に生成するためのフラグは FLAG_ACTIVITY_NEW_TASK である。FLAG_ACTIVITY_NEW_TASK が設定されると、呼び出された Activity のタスクがバックグラウンドあるいはフォアグラウンド上に存在しない場合に、新規にタスクが生成される。FLAG_ACTIVITY_MULTIPLE_TASK は FLAG_ACTIVITY_NEW_TASK と同時に設定することもできる。この場合には、タスクが必ず新規生成される。どちらの設定もタスクを生成する可能性があるため、センシティブな情報を扱う Intent には設定しないようにすべきである。

Intent の送信例

```
Intent intent = new Intent();

// ★ 4.1.1.1 - ポイント 6★ Activity に送信する Intent には、フラグ FLAG_ACTIVITY_NEW_TASK を設定しない

intent.setClass(this, PrivateActivity.class);
intent.putExtra("PARAM", "センシティブな情報");

startActivityForResult(intent, REQUEST_CODE);
```

なお、Activity に送信する Intent に FLAG_ACTIVITY_EXCLUDE_FROM_RECENTS フラグを明示的に設定することで、タスクが生成されたとしてもその内容が読み取られないようにできると考えるかもしれない。しかしながら、この方法を用いても送信された Intent の内容を読み取ることが可能である。したがって、FLAG_ACTIVITY_NEW_TASK の使用は避けるべきである。

「4.1.3.1. *exported* 設定と *intent-filter* 設定の組み合わせ *(Activity* の場合*)*」および「4.1.3.3. ルート *Activity* について」も参照すること。

4.1.2.5 受信 Intent の安全性を確認する（必須）

Activity のタイプによって若干リスクは異なるが、受信 Intent のデータを処理する際には、まず受信 Intent の安全性を確認しなければならない。

公開 Activity は不特定多数のアプリから Intent を受け取るため、マルウェアの攻撃 Intent を受け取る可能性がある。非公開 Activity は他のアプリから Intent を直接受け取ることはない。しかし同一アプリ内の公開 Activity が他のアプリから受け取った Intent のデータを非公開 Activity に転送することがあるため、受信 Intent を無条件に安全であると考えてはならない。パートナー限定 Activity や自社限定 Activity はその中間のリスクであるため、やはり受信 Intent の安全性を確認する必要がある。

「3.2. 入力データの安全性を確認する」を参照すること。

4.1.2.6 独自定義 Signature Permission は、自社アプリが定義したことを確認して利用する（必須）

自社アプリだけから利用できる自社限定 Activity を作る場合、独自定義 Signature Permission により保護しなければならない。AndroidManifest.xml での Permission 定義、Permission 要求宣言だけでは保護が不十分であるため、「5.2. *Permission* と *Protection Level*」の「5.2.1.2. 独自定義の *Signature Permission* で自社アプリ連携する方法」を参照すること。

4.1.2.7 結果情報を返す場合には、返送先アプリからの結果情報漏洩に注意する（必須）

Activity のタイプによって、setResult() を用いて結果情報を返送する際の返送先アプリの信用度が異なる。公開 Activity が結果情報を返送する場合、結果返送先アプリがマルウェアである可能性があり、結果情報が悪意を持って使われる危険性がある。非公開 Activity や自社限定 Activity の場合は、結果返送先は自社アプリであるため結果情報の扱いをあまり心配する必要はない。パートナー限定 Activity の場合はその中間に位置する。

このように Activity から結果情報を返す場合には、返送先アプリからの結果情報の漏洩に配慮しなければならない。

結果情報を返送する場合の例

```
public void onReturnResultClick(View view) {

    // ★ 4.1.1.3 - ポイント 6★ パートナーアプリに開示してよい情報に限り返送してよい
    Intent intent = new Intent();
    intent.putExtra("RESULT", "パートナーアプリに開示してよい情報");
    setResult(RESULT_OK, intent);
    finish();
}
```

4.1.2.8 利用先 **Activity** が固定できる場合は明示的 **Intent** で **Activity** を利用する（必須）

暗黙的 Intent により Activity を利用すると、最終的にどの Activity に Intent が送信されるかは Android OS 任せになってしまう。もしマルウェアに Intent が送信されてしまうと情報漏洩が生じる。一方、明示的 Intent により Activity を利用すると、指定した Activity 以外が Intent を受信することはなく比較的安全である。

処理を任せるアプリ（の Activity）をユーザーに選択させるなど、利用先 Activity を実行時に決定したい場合を除けば、利用先 Activity はあらかじめ特定できる。このような Activity を利用する場合には明示的 Intent を利用すべきである。

同一アプリ内の Activity を明示的 Intent で利用する

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

他のアプリの公開 Activity を明示的 Intent で利用する

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

ただし他のアプリの公開 Activity を明示的 Intent で利用した場合も、相手先 Activity を含むアプリがマルウェアである可能性がある。宛先をパッケージ名で限定したとしても、相手先アプリが実は本物アプリと同じパッケージ名を持つ偽物アプリである可能性があるからだ。このようなリスクを排除したい場合は、パートナー限定 Activity や自社限定 Activity の使用を検討する必要がある。

「4.1.3.1. *exported* 設定と *intent-filter* 設定の組み合わせ *(Activity* の場合*)*」も参照すること。

4.1.2.9 利用先 **Activity** からの戻り **Intent** の安全性を確認する（必須）

Activity のタイプによって若干リスクは異なるが、戻り値として受信した Intent のデータを処理する際には、まず受信 Intent の安全性を確認しなければならない。

利用先 Activity が公開 Activity の場合、不特定のアプリから戻り Intent を受け取るため、マルウェアの攻撃 Intent を受け取る可能性がある。利用先 Activity が非公開 Activity の場合、同一アプリ内から戻り Intent を受け取るのでリスクはないように考えがちだが、他のアプリから受け取った Intent のデータを間接的に戻り値として転送することがあるため、受信 Intent を無条件に安全であると考えてはならない。利用先 Activity がパートナー限定 Activity や自社限定 Activity の場合、その中間のリスクであるため、やはり受信 Intent の安全性を確認する必要がある。

「3.2. 入力データの安全性を確認する」を参照すること。

4.1.2.10 他社の特定アプリと連携する場合は利用先 **Activity** を確認する（必須）

他社の特定アプリと連携する場合にはホワイトリストによる確認方法がある。自アプリ内に利用先アプリの証明書ハッシュを予め保持しておく。利用先の証明書ハッシュと保持している証明書ハッシュが一致するかを確認することで、なりすましアプリに Intent を発行することを防ぐことができる。具体的な実装方法についてはサンプルコードセクション「4.1.1.3. パートナー限定 *Activity* を作る・利用する」を参照すること。また、技術的な詳細に関しては「4.1.3.2. 利用元アプリを確認する」を参照すること。

4.1.2.11 資産を二次的に提供する場合には、その資産の従来の保護水準を維持する（必須）

Permission により保護されている情報資産および機能資産を他のアプリに二次的に提供する場合には、提供先アプリに対して同一の Permission を要求するなどして、その保護水準を維持しなければならない。Android の Permission セキュリティモデルでは、保護された資産に対するアプリからの直接アクセスについてのみ権限管理を行う。この仕様上の特性により、アプリに取得された資産がさらに他のアプリに、保護のために必要な Permission を要求することなく提供される可能性がある。このことは Permission を再委譲 (Redelegation) していることと実質的に等価なので、Permission の再委譲問題と呼ばれる。「5.2.3.4. *Permission* の再委譲問題」を参照すること。

4.1.2.12 センシティブな情報はできる限り送らない （推奨）

不特定多数のアプリと連携する場合にはセンシティブな情報を送ってはならない。特定のアプリと連携する場合においても、意図しないアプリに Intent を発行してしまった場合や第三者による Intent の盗聴などで情報が漏洩してしまうリスクがある。「4.1.3.4. *Activity* 利用時のログ出力について」を参照すること。

センシティブな情報を Activity に送付する場合、その情報の漏洩リスクを検討しなければならない。公開 Activity に送付した情報は必ず漏洩すると考えなければならない。またパートナー限定 Activity や自社限定 Activity に送付した情報もそれら Activity の実装に依存して情報漏洩リスクの大小がある。非公開 Activity に送付する情報に至っても、Intent の data に含めた情報は LogCat 経由で漏洩するリスクがある。Intent の extras は LogCat に出力されないので、センシティブな情報は extras で送付するとよい。

センシティブな情報はできるだけ送付しないように工夫すべきである。送付する場合も、利用先 Activity は信頼できる Activity に限定し、Intent の情報が LogCat へ漏洩しないように配慮しなければならない。

また、ルート Activity にはセンシティブな情報を送ってはならない。ルート Activity とは、タスクが生成された時に最初に呼び出された Activity のことである。例えば、ランチャーから起動された Activity は常にルート Activity である。

ルート Activity に関しての詳細は、「4.1.3.3. ルート *Activity* について」も参照すること。

4.1.3 アドバンスト

4.1.3.1 exported 設定と intent-filter 設定の組み合わせ (Activity の場合)

このガイド文書では、Activity の用途から非公開 Activity、公開 Activity、パートナー限定 Activity、自社限定 Activity の 4 タイプの Activity について実装方法を述べている。各タイプに許されている AndroidManifest.xml の exported 属性と intent-filter 要素の組み合わせを次の表にまとめた。作ろうとしている Activity のタイプと exported 属性および intent-filter 要素の対応が正しいことを確認すること。

表 4.1.1　exported 設定と intent-filter 設定の組み合わせ

exported 属性の値	intent-filter 定義がある	intent-filter 定義がない
true	公開	公開、パートナー限定、自社限定
false	（使用禁止）	非公開
無指定	（使用禁止）	（使用禁止）

Activity の exported 属性が無指定である場合にその Activity が公開されるか非公開となるかは、intent-filter の定義の有無により決まるが[*4]、本ガイドでは Activity の exported 属性を「無指定」にすることを禁止している。前述のような API

[*4] intent-filter が定義されていれば公開 Activity、定義されていなければ非公開 Activity となる。https://developer.android.com/guide/topics/manifest/

のデフォルトの挙動に頼る実装をすることは避けるべきであり、exported 属性のようなセキュリティ上重要な設定を明示的に有効化する手段があるのであればそれを利用すべきであると考えられるためである。

exported 属性の値で「intent-filter 定義がある」&「exported=”false”」を使用禁止にしているのは、Android の振る舞いに抜け穴があり、Intent Filter の性質上、意図せず他アプリの Activity を呼び出してしまう場合が存在するためである。以下の 2 つの図は、その説明のためのものである。図 4.1.4 は、同一アプリ内からしか非公開 Activity(アプリ A）を暗黙的 Intent で呼び出せない正常な動作の例である。Intent-filter(図中 action=”X”) を定義しているのが、アプリ A しかいないので意図通りの動きとなっている。

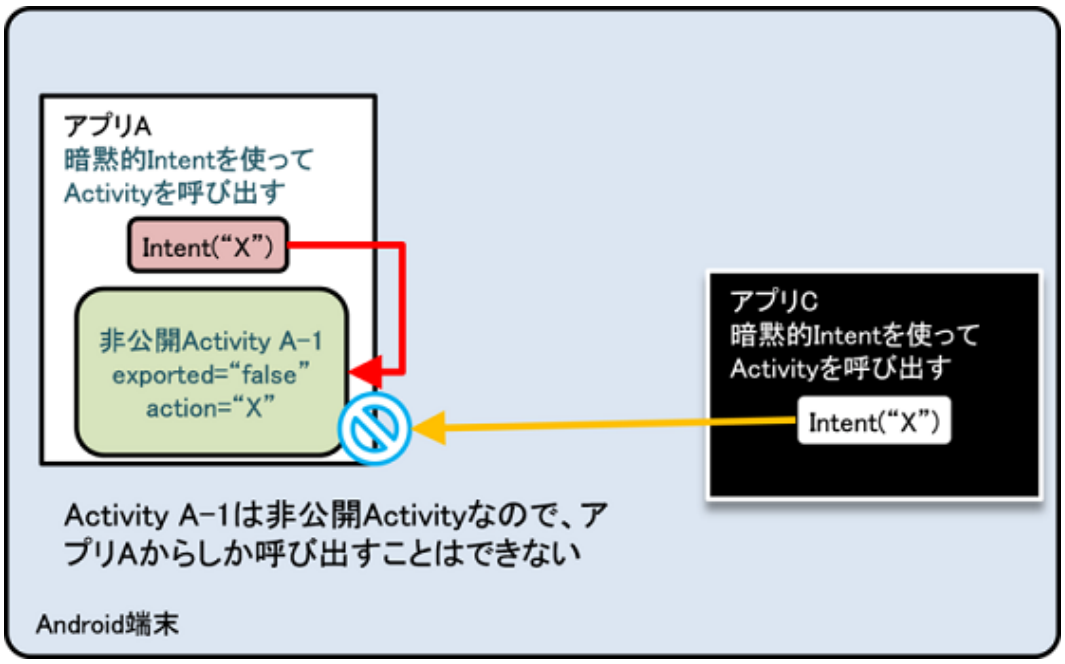

図 4.1.4　正常な動作の例

図 4.1.5 は、アプリ A に加えてアプリ B でも同じ intent-filter(図中 action=”X”) を定義している場合である。図 4.1.5 では、アプリ A が暗黙的 Intent を送信して同一アプリ内の非公開 Activity を呼び出そうとするが、「アプリケーションを選択」ダイアログが表示され、ユーザーの選択によって公開 Activity(B-1) が呼び出されてしまう例を示している[*5]。これにより他アプリに対してセンシティブな情報を送信したり、意図せぬ戻り値を受け取る可能性が生じてしまう。

activity-element.html#exported を参照のこと。

[*5] Android 8.0(API Level 26) 以降の端末では「アプリケーションを選択」ダイアログは表示されず、図中アプリ B の公開 Activity(B-1) に自動遷移することが確認されている。このことからも、Intent Filter を持つ非公開 Activity を暗黙的 Intent によって起動するのは禁止すべきである。

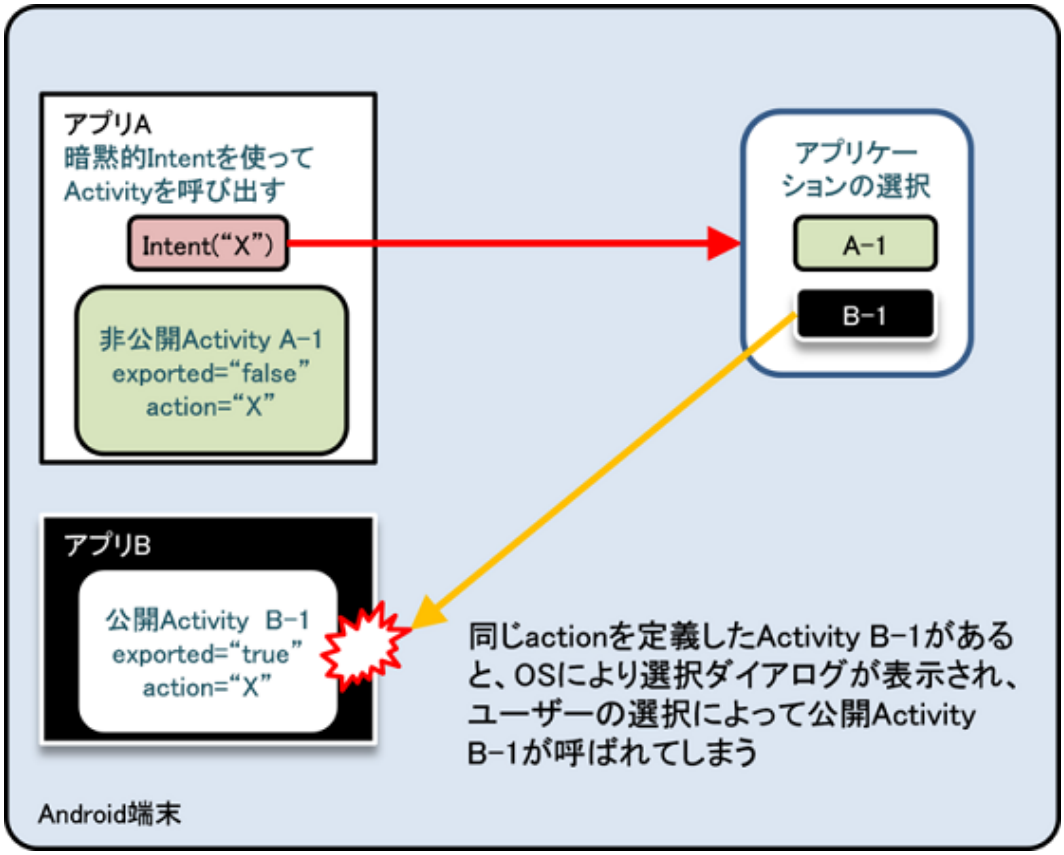

図 4.1.5　意図しない動作の例

このように、Intent Filter を用いた非公開 Activity の暗黙的 Intent 呼び出しは、意図せぬアプリとの情報のやり取りを許してしまうので行ってはならない。なお、この挙動はアプリ A、アプリ B のインストール順序には依存しないことを確認している。

なお、Android 12 以降、exported 設定は必須となっている。詳細は、4.7.3.1. コンポーネント公開制御と *Intent* 送信の制限」を参照すること。

4.1.3.2 利用元アプリを確認する

ここではパートナー限定 Activity の実装に関する技術情報を解説する。パートナー限定 Activity はホワイトリストに登録された特定のアプリからのアクセスを許可し、それ以外のアプリからはアクセスを拒否する Activity である。自社以外のアプリもアクセス許可対象となるため、Signature Permission による防御手法は利用できない。

基本的な考え方は、パートナー限定 Activity の利用元アプリの身元を確認し、ホワイトリストに登録されたアプリであればサービスを提供する、登録されていないアプリであればサービスを提供しないというものである。利用元アプリの身元確認は、利用元アプリが持つ証明書を取得し、その証明書のハッシュ値をホワイトリストのハッシュ値と比較することで行う。

ここでわざわざ利用元アプリの「証明書」を取得せずとも、利用元アプリの「パッケージ名」との比較で十分ではないか？ と疑問を持たれた方もいるかと思う。しかしパッケージ名は任意に指定できるため他のアプリへの成りすましが簡単である。成りすまし可能なパラメータは身元確認用には使えない。一方、アプリの持つ証明書であれば身元確認に使うことができる。証明書に対応する署名用の開発者鍵は本物のアプリ開発者しか持っていないため、第三者が同じ証明書を持ち、尚且つ署名検証が成功するアプリを作成することはできないからだ。ホワイトリストはアクセスを許可したいアプリの証明書データを丸ごと保持してもよいが、サンプルコードではホワイトリストのデータサイズを小さくするために証明書データの SHA-256 ハッシュ値を保持することにしている。

この方法には次の二つの制約条件がある。

- 利用元アプリにおいて startActivity() ではなく startActivityForResult() を使用しなければならない
- 利用元アプリにおいて Activity 以外から呼び出すことはできない

2 つ目の制約事項はいわば 1 つ目の制約事項の結果として課される制約であるので、厳密には 1 つの同じ制約と言える。この制約は呼び出し元アプリのパッケージ名を取得する Activity.getCallingPackage() の制約により生じている。Activity.getCallingPackage() は startActivityForResult() で呼び出された場合にのみ利用元アプリのパッケージ名を返すが、残念ながら startActivity() で呼び出された場合には null を返す仕様となっている。そのためここで紹介する方法は必ず利用元アプリが、たとえ戻り値が不要であったとしても、startActivityForResult() を使わなければならないという制約がある。さらに startActivityForResult() は Activity クラスでしか使えないため、利用元は Activity に限定されるという制約もある。

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 {

    // ★ 4.1.1.3 - ポイント 4★ 利用元アプリの証明書がホワイトリストに登録されていることを確認する
    private static PkgCertWhitelists sWhitelists = null;
```

(次のページに続く)

(前のページからの続き)

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

        // パートナーアプリ org.jssec.android.activity.partneruser の証明書ハッシュ値を登録
        sWhitelists.add("org.jssec.android.activity.partneruser", isdebug ?
            // debug.keystore の"androiddebugkey"の証明書ハッシュ値
            "0EFB7236 328348A9 89718BAD DF57F544 D5CCB4AE B9DB34BC 1E29DD26 F77C8255" :
            // keystore の"partner key"の証明書ハッシュ値
            "1F039BB5 7861C27A 3916C778 8E78CE00 690B3974 3EB8259F E2627B8D 4C0EC35A");

        // 以下同様に他のパートナーアプリを登録...
    }
    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);

        // ★ 4.1.1.3 - ポイント 4★ 利用元アプリの証明書がホワイトリストに登録されていることを確認する
        if (!checkPartner(this, getCallingActivity().getPackageName())) {
            Toast.makeText(this, "利用元アプリはパートナーアプリではない。",
                           Toast.LENGTH_LONG).show();
            finish();
            return;
        }

        // ★ 4.1.1.3 - ポイント 5★ パートナーアプリからの Intent であっても、受信 Intent の安全性を確
認する
        // サンプルにつき割愛。「3.2 入力データの安全性を確認する」を参照。
        Toast.makeText(this, "パートナーアプリからアクセスあり", Toast.LENGTH_LONG).show();
    }

    public void onReturnResultClick(View view) {

        // ★ 4.1.1.3 - ポイント 6★ パートナーアプリに開示してよい情報に限り返送してよい
        Intent intent = new Intent();
        intent.putExtra("RESULT", "パートナーアプリに開示してよい情報");
        setResult(RESULT_OK, intent);
        finish();
    }
```

(次のページに続く)

(前のページからの続き)

```
}
```

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 は 32 バイト
        sha256 = sha256.toUpperCase();
        if (sha256.replaceAll("[0-9A-F]+", "").length() != 0)
            return false;    // 0-9A-F 以外の文字がある

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

(次のページに続く)

(前のページからの続き)

```
        return true;
    }

    public boolean test(Context ctx, String pkgname) {
        // pkgname に対応する正解のハッシュ値を取得する
        String correctHash = mWhitelists.get(pkgname);

        // pkgname の実際のハッシュ値と正解のハッシュ値を比較する
        if (Build.VERSION.SDK_INT >= 28) {
            // ★ API Level >= 28 では Package Manager の API で直接検証が可能
            PackageManager pm = ctx.getPackageManager();
            return pm.hasSigningCertificate(pkgname,
                                            Utils.hex2Bytes(correctHash),
                                            CERT_INPUT_SHA256);
        } else {
            // API Level < 28 の場合は PkgCert の機能を利用する
            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;
```

(次のページに続く)

(前のページからの続き)

```
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);
            if (pkginfo.signatures.length != 1) return null;  // 複数署名は扱わない
            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();
    }
```

(次のページに続く)

(前のページからの続き)

```
}
```

4.1.3.3 ルート **Activity** について

ルート Activity とは、タスクの起点となる Activity のことである。タスクが生成された時に最初に起動された Activity を指す。たとえば、デフォルト設定の Activity がランチャーから起動された場合、その Activity がルート Activity となる。

Android 5.0（API Level 21）以降の仕様変更により、ルート Activity に送信された Intent の内容が他のアプリから読み取られるリスクは解消されている。そのため、従来必要とされていた「ルート Activity へセンシティブな情報を送信しない」といったセキュリティ対策は、Android 7 以降では不要となった。

ただし、タスク管理の複雑化や予期せぬ挙動を防ぐため、アプリ設計上は以下の点に注意することが望ましい。

- taskAffinity は特別な理由がない限り指定しない
- launchMode は特別な理由がない限り指定しない（デフォルトの standard を推奨する）
- Activity に送信する Intent に FLAG_ACTIVITY_NEW_TASK を安易に設定しない

Activity の起動モードについて

Activity の起動モードは、AndroidManifest.xml の android:launchMode 属性で設定できる。記述しない場合は”standard” とみなされる。また、起動モードは Intent に設定するフラグによっても影響を受ける。たとえば、FLAG_ACTIVITY_NEW_TASK フラグは、Activity を”singleTask” モードで起動する効果を持つ。

Activity の起動モードは次の 5 種類である。

表 4.1.2　Activity の起動モード一覧

起動モード	説明
standard	デフォルトのモードであり、呼び出されるたびに新しいインスタンスが生成される。呼び出し側のタスクに所属する。
singleTop	standard とほぼ同じだが、最前面に同じ Activity がある場合は新しいインスタンスを生成せず、既存のインスタンスが再利用される。
singleTask	Activity のアフィニティと一致するタスクが存在しない場合、新規タスクが生成され、そのルート Activity となる。既に存在する場合は再利用される。
singleInstance	singleTask に似ているが、生成されたタスクにはこの Activity のみが所属できる。常にルート Activity となる。
singleInstancePerTask	Android 12 以降で利用可能。タスクごとにルート Activity としてのみ存在できる。

singleTask や singleInstance で起動された Activity は、タスクのルートとなる場合がある。しかし、Android 7 以降では、これらのモードを利用しても他アプリから Intent 内容が読み取られる心配はない。ただし、タスク構造が複雑になることで、アプリの挙動に影響を与える可能性があるため、特殊な用途がない限り、standard モードを基本とすることが望ましい。

Android 7 以降では、ルート Activity に送信された Intent が他アプリから読み取られるリスクは存在しない。そのため、従来のセキュリティ対策は不要となったが、アプリの設計やユーザー体験を考慮し、taskAffinity や launchMode の不要な指定、FLAG_ACTIVITY_NEW_TASK の安易な利用は避けることが推奨される。

4.1.3.4 **Activity** 利用時のログ出力について

Activity を利用する際に ActivityManager が Intent の内容を LogCat に出力する。以下の内容は LogCat に出力されるため、センシティブな情報が含まれないように注意すべきだ。

- 利用先パッケージ名
- 利用先クラス名
- Intent#setData() で設定した URI

例えば、メール送信する場合、URI にメールアドレスを指定して Intent を発行するとメールアドレスが LogCat に出力されてしまう。Intent#putExtra() で設定した値は LogCat に出力されないため、Extras に設定して送るようにした方が良い。

次のようにメール送信すると LogCat にメールアドレスが表示されてしまう

```
MainActivity.java
    // URI は LogCat に出力される
    Uri uri = Uri.parse("mailto:test@gmail.com");
    Intent intent = new Intent(Intent.ACTION_SENDTO, uri);
    startActivity(intent);
```

次のように Extras を使用すると LogCat にメールアドレスが表示されなくなる

```
MainActivity.java
    // Extra に設定した内容は LogCat に出力されない
    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);
```

ただし、ActivityManager#getRecentTasks() によって Intent の Extras を他のアプリから直接読める場合があるので、注意すること。詳しくは「4.1.2.2. *taskAffinity* を指定しない（必須）」、「4.1.2.3. *launchMode* を指定しない（必須）」および「4.1.2.4. *Activity* に送信する *Intent* には *FLAG_ACTIVITY_NEW_TASK* を設定しない（必須）」を参照のこと。

4.1.3.5 設定画面の **Fragment Injection** 対策について

PreferenceActivity および android.preference パッケージは、Android 10（API レベル 29）以降で非推奨となっており、今後の新規開発や保守では AndroidX の PreferenceFragmentCompat 等への移行が推奨されている。

PreferenceActivity を継承したクラスが公開 Activity となっている場合、Fragment Injection[*6] と呼ばれる問題が発生する可能性があるが、現行の推奨実装では PreferenceFragmentCompat を利用することでこのリスクを回避できる。

PreferenceFragmentCompat を利用した場合、Fragment Injection のリスクは原則的に存在しない。設定画面は AppCompatActivity ＋ PreferenceFragmentCompat で構成する。

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

（次のページに続く）

[*6] Fragment Injection の詳細は以下の URL を参照のこと https://securityintelligence.com/new-vulnerability-android-framework-fragment-injection/

(前のページからの続き)

```
        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)
        }
    }
}
```

やむを得ず PreferenceActivity を使い続ける場合（レガシー端末対応など）は、Fragment Injection 脆弱性への対策が必須となる。isValidFragment() を必ず override し、許可する Fragment だけを明示的に指定すること。さらに、マニフェストで android:exported=”false” を設定し、外部からの不要な呼び出しを防ぐようにする。

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

難読化を行う場合は、クラス名が変わらないように ProGuard などで-keepnames オプションを設定しておく必要がある。なお、Google Play では Fragment Injection 脆弱性が残ったままの PreferenceActivity を含むアプリは、新規公開やアップデートがブロックされる場合がある。保守性やセキュリティの観点からも、PreferenceActivity の利用は避け、できるだけ早く AndroidX の Preference ライブラリへ移行することを強く推奨する。

4.1.3.6 Autofill フレームワークについて

Autofill フレームワークは Android 8.0(API Level 26) で追加された仕組みである。この仕組みを利用することで、ユーザー名、パスワード、住所、電話番号、クレジットカード情報等が入力されたときにそれらを保存しておき、再度必要になった時に取り出してアプリに自動入力 (Autofill) する機能を実現することができる。ユーザーの入力負荷を軽減することができる便利な仕組みであるが、あるアプリが扱うパスワードやクレジットカード情報等のセンシティブな情報を他のアプリ（Autofill service）に渡すことを想定しており、取り扱いには十分に気をつける必要がある。

仕組み (概要)

2 つのコンポーネント

以下に、Autofill フレームワークに登場する 2 つのコンポーネント[*7] の概要を示す。

- Autofill の対象となるアプリ（利用アプリ）:
  - View の情報（テキストおよび属性) を Autofill service に渡したり、Autofill service から Autofill に必要な情

[*7] 「利用アプリ」と「Autofill service」は、それぞれ同じパッケージ (APK ファイル) であることも、別パッケージであることもあり得る。

報を提供されたりする。

- Activity を持つすべてのアプリが利用アプリとなる（Foreground 時)。
- 利用アプリが持つすべての View が Autofill の対象になる可能性がある。View 単位で明示的に Autofill の対象外とすることも可能。
- Autofill 機能の利用を同一パッケージ内の Autofill service に限定することも可能。

- Autofill service を提供するサービス（Autofill service）:
  - アプリから渡された View の情報を保存したり（ユーザーの許可が必要)、View に Autofill するための情報（候補リスト）をアプリに提供したりする。
  - 保存対象の View は Autofill service が決定する（Autofill フレームワークはデフォルトで Activity に含まれるすべての View の情報を Autofill service に渡す)。
  - 3rd Party 製の Autofill service も作成できる。
  - 端末内に複数存在することが可能でユーザーにより「設定」から選択された Service のみ有効になる（「なし」も選択可能)。
  - Service が、扱うユーザー情報を保護するためにパスワード入力等によってユーザー認証をするための UI を持つことも可能。

Autofill フレームワークの処理フロー

図 4.1.6 は Autofill 時の Autofill 関連コンポーネント間の処理フローを示している。利用アプリの View のフォーカス移動等を契機に Autoill フレームワークを介して View の情報（主に View の親子関係や個々の属性）が「設定」で選択された Autofill service に渡る。Autofill service は渡された情報を元に Autofill に必要な情報（候補リスト）を DB から取り出し、フレームワークに返信する。フレームワークは候補リストをユーザーに提示し、ユーザーが選択したデータによりアプリで Autofill が行われる。

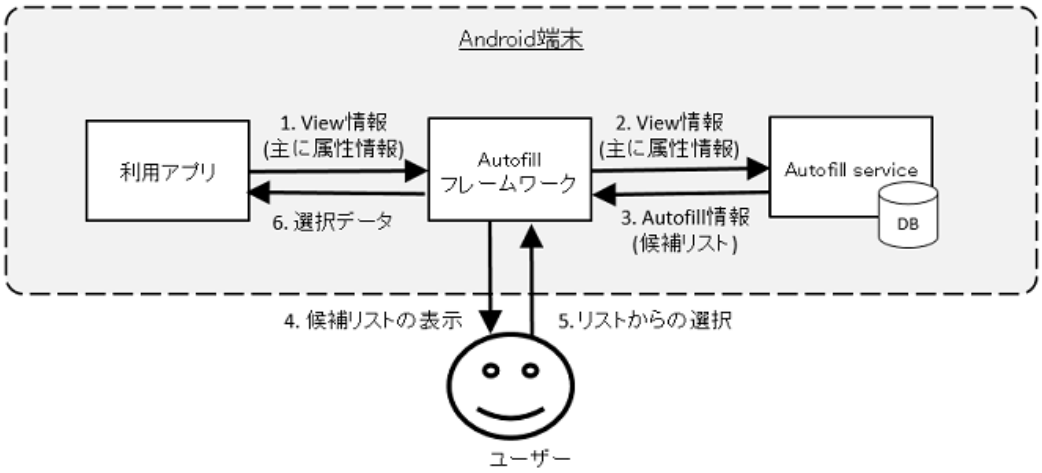

図 4.1.6　Autofill 時のコンポーネント間の処理フロー

一方、図 4.1.7 は Autofill によるユーザーデータ保存時の処理フローを示している。AutofillManager#commit() の呼び出しや Activity の終了を契機に、Autofill した View の値に変更があり、かつ、Autofill フレームワークが表示する保存許可ダイアログに対してユーザーが許可した場合、View の情報（テキスト含む）が Autofill フレームワークを介して「設定」で選択された Autofill service に渡され、Autofill service が View の情報を DB に保存して一連の処理が完了となる。

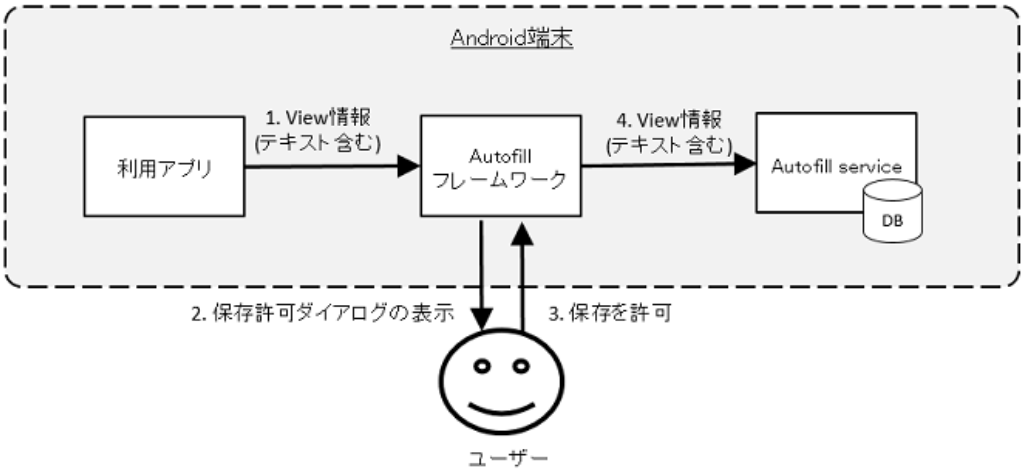

図 4.1.7　ユーザーデータ保存時のコンポーネント間の処理フロー

Autofill 利用アプリにおけるセキュリティ上の懸案

「仕組み（概要）」の項で示した通り Autofill フレームワークにおけるセキュリティモデルでは、ユーザーが「設定」で安全な Autofill service を選択し、保存時にどのデータをどの Autofill service に渡してもよいか適切に判断できることを前提としている。

ところが、ユーザーがうっかり安全でない Autofill service を選択したり、Autofill service に渡すべきでないセンシティブな情報の保存を許可してしまったりする可能性がある。以下に、この場合に起きうる被害について考察する。

保存時、ユーザーが Autofill service を選択し、保存許可ダイアログに対して許可した場合、利用アプリで表示されている Activity に含まれるすべての View の情報が Autofill service に渡る可能性がある。ここで、Autofill service がマルウェアの場合や、Autofill service に View の情報を外部ストレージや安全でないクラウドサービスに保存する等のセキュリティ上の問題があった場合には、利用アプリで扱う情報の漏洩につながってしまうリスクが考えられる。

一方、Autofill 時、ユーザーが Autofill service としてマルウェアを選択してしまっていた場合、マルウェアが送信した値を入力してしまう可能性がある。ここで、アプリやアプリのデータを送信した先のクラウドサービスが入力データの安全性を十分に確認していなかった場合、情報漏洩やアプリ／サービスの停止等につながってしまうリスクが考えられる。

なお、「2 つのコンポーネント」で書いたように、Activity を持つアプリが自動的に Autofill の対象になるため、Activity を持つアプリのすべての開発者は上記のリスクを考慮して設計や実装を行う必要がある。以下に、上記のリスクに対する対策案を示すが、「3.1.3. 資産分類と保護施策」なども参考にして、アプリに必要な対策を検討した上で、適用することをお勧めする。

リスクに対する対策-*1*

前述のように、Autofill フレームワークでは基本的にユーザーの裁量によってセキュリティが担保されている。そのためアプリでできる対策は限られているが、View に対して importantForAutofill 属性で”no” 等を指定して Autofill service に View の情報を渡さないようにする（Autofill の対象外とする）ことで、ユーザーが適切な選択や許可をできなかった場合（マルウェアを Autofill service として利用するように選択する等）でも、上記の懸案を軽減することができる。[*8]

- レイアウト XML の importantForAutofill 属性を指定する
- View#setImportantForAutofill() を呼び出す

[*8] ユーザーが意図的に Autofill 機能を利用した場合など、本対策でも上記の懸案を回避できないことがある。「リスクに対する対策-2」を実施することでこのような場合にも対応することができる。importantForAutofill 属性は、以下のいずれかの方法によって指定することができる。

以下に指定可能な値を示す。指定する範囲によって適切な値を使うこと。特に、”no” を指定した場合、指定した View は Autofill の対象外になるが、子供は Autofill の対象になることに注意すること。デフォルト値は、”auto” となっている。

表 4.1.3　Autofill の対象になるか

値 定数名	指定した **View**	子供の **View**
”auto” IMPORTANT_FOR_AUTOFILL_AUTO	自動[9]	自動[9]
”no” IMPORTANT_FOR_AUTOFILL_NO	対象外	対象
”noExcludeDescendants” IMPORTANT_FOR_AUTOFILL_NO_EXCLUDE_DESCENDANTS	対象外	対象外
”yes” IMPORTANT_FOR_AUTOFILL_YES	対象	対象
”yesExcludeDescendants” IMPORTANT_FOR_AUTOFILL_YES_EXCLUDE_DESCENDANTS	対象	対象外

また、AutofillManager#hasEnabledAutofillServices() を利用して、Autofill 機能の利用を同一パッケージ内の Autofill service に限定することも可能である。

以下に、「設定」で同一パッケージ内の Autofill service を利用するように設定されている場合のみ、Activity の全ての View を Autofill の対象にする（実際に Autofill の対象になるかは Autofill service 次第）場合の例を示す。個別の View に対して View#setImportantForAutofill() を呼び出すことも可能である。

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;
```

(次のページに続く)

[9] Autofill フレームワークが決定

(前のページからの続き)

```
import android.service.autofill.UserData;
import android.app.Activity;
import android.util.Log;
import android.view.View;
import android.view.autofill.AutofillManager;
import android.widget.EditText;
import android.widget.TextView;

import org.jssec.android.autofillframework.R;

import java.util.Iterator;
import java.util.List;

public class DisableForOtherServiceActivity extends Activity {
    private String TAG = "AutofillSample";
    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();
            }
        });
        //この Activity ではフローティングツールバーの対応をしていないため、
        // 「自動入力」の選択で Autofill 機能が利用可能になります。
    }
```

(次のページに続く)

(前のページからの続き)

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

    @Override
    protected void onResume() {
        super.onResume();
        updateAutofillStatus();
        View rootView = this.getWindow().getDecorView();
        if (!mIsAutofillEnabled) {
            //同一パッケージ内の Autofill service を利用しない場合は、全ての View を Autofill の対象外
にする
            rootView.setImportantForAutofill(View.IMPORTANT_FOR_AUTOFILL_NO_EXCLUDE_
↪DESCENDANTS);
        } else {
            //同一パッケージ内の Autofill service を利用する場合は、全ての View を Autofill の対象にす
る
            //特定の View に対して View#setImportantForAutofill() を呼び出すことも可能
            rootView.setImportantForAutofill(View.IMPORTANT_FOR_AUTOFILL_AUTO);
        }
        testFieldClassification();
    }
    private void login() {
        String username = mUsernameEditText.getText().toString();
        String password = mPasswordEditText.getText().toString();

        //View から取得したデータの安全性を確認する
        if (!Util.validateUsername(username) || !Util.validatePassword(password)) {
            //適切なエラー処理をする
        }

        //サーバーに username, password を送信

        finish();
    }

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

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

(次のページに続く)

(前のページからの続き)

```
        mIsAutofillEnabled = mgr.hasEnabledAutofillServices();

        TextView statusView = (TextView) findViewById(R.id.label_autofill_status);
        String status = "自社の autofill service が--です";
        if (mIsAutofillEnabled) {
            status = "同一パッケージ内の autofill service が有効です";
        } else {
            status = "同一パッケージ内の autofill service が無効です";
        }
        statusView.setText(status);
    }
    private void testFieldClassification() {
        AutofillManager mgr = getSystemService(AutofillManager.class);
        List<String> algorithms = mgr.getAvailableFieldClassificationAlgorithms();
        Log.d(TAG, "algorithms num=" + algorithms.size());
        Iterator it = algorithms.iterator();
        while (it.hasNext()) {
            Log.d(TAG, "algorithms=" + (String)it.next());
        }
        String defaultAlgorithm = mgr.getDefaultFieldClassificationAlgorithm();
        Log.d(TAG, "default algorithms=" + defaultAlgorithm);

        boolean isFieldClassificationEnabled = mgr.isFieldClassificationEnabled();
        Log.d(TAG, "isFieldClassificationEnabled=" + isFieldClassificationEnabled);

        UserData userData = mgr.getUserData();
        if (userData != null) {
            Log.d(TAG, "userData fcAlgorithms=" +
                    userData.getFieldClassificationAlgorithm());
        } else {
            Log.d(TAG, "userData is null");
        }
    }
}
```

リスクに対する対策-*2*

アプリで「リスクに対する対策-*1*」を施した場合でも、ユーザーが View の長押しでフローティングツールバーなどを表示させて「自動入力」を選択すると、強制的に Autofill を利用できてしまう。この場合、importantForAutofill 属性で”no”等を指定した View を含む全ての View の情報が Autofill Service に渡ることになる。

「リスクに対する対策-*1*」に加えて、フローティングツールバーなどのメニューから「自動入力」を削除することで、上記のような場合でも、情報漏えいのリスクを回避することができる。

以下にサンプルコードを示す。

DisableAutofillActivity.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.autofillframework.autofillapp;

import android.os.Bundle;
import android.app.Activity;
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 Activity {

    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);
```

(次のページに続く)

(前のページからの続き)

```
        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);
                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) {
            }
        };

        //フローティングツールバーから「自動入力」を削除する
        setMenu();
    }

    void setMenu() {
        if (mActionModeCallback == null) {
            return;
        }
        //Activity に含まれる全ての編集可能な TextView についてセットすること
        mUsernameEditText.setCustomInsertionActionModeCallback(mActionModeCallback);
```

(次のページに続く)

(前のページからの続き)

```
        mPasswordEditText.setCustomInsertionActionModeCallback(mActionModeCallback);
    }

    //Menu の階層を巡って「自動入力」を削除する
    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();

        //View から取得したデータの安全性を確認する
        if (!Util.validateUsername(username) || Util.validatePassword(password)) {
            //適切なエラー処理をする
        }

        //サーバーに username, password を送信

        finish();
    }

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

リスクに対する対策-*3*

Android 9.0 (API Level 28) で、現在有効となっている Autofill Service のコンポーネントが何であるかを AutofillManager#getAutofillServiceComponentName() によって知ることができるようになった。これを用いてパッケージ名を取得し、自身のアプリが信頼を置いている Autofill Service かどうかを確認できる。

この場合、先に「4.1.3.2. 利用元アプリを確認する」で述べた通りパッケージ名は成りすましが可能なため、これのみで身元確認をするのは推奨できない。4.1.3.2. に掲載の例と同じく、パッケージ名から Autofill Service の証明書を取得し、それをあらかじめホワイトリストに登録されているものと一致するかどうかによって身元確認を行うべきである。この

方法の詳細については 4.1.3.2. に詳しいのでそちらを参照されたい。

以下に、あらかじめホワイトリストに登録されている Autofill Service が有効になっている場合のみ、Activity の全ての View を Autofill の対象にする場合の例を示す。

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;

public class EnableOnlyWhitelistedServiceActivity extends Activity {
    private static PkgCertWhitelists sWhitelists = null;
    private static void buildWhitelists(Context context) {
        sWhitelists = new PkgCertWhitelists();
        // 信頼する Autofill Service の証明書ハッシュ値を登録
        sWhitelists.add("com.google.android.gms",
            "1975B2F17177BC89A5DFF31F9E64A6CAE281A53DC1D1D59B1D147FE1C82AFA00");
        // 以下同様に他の信頼する Autofill Service を登録...
    }
```

(次のページに続く)

(前のページからの続き)

```
    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();
                }
            });
        //この Activity ではフローティングツールバーの対応をしていないため、
        // 「自動入力」の選択で Autofill 機能が利用可能になります。
    }

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

    @Override
    protected void onResume() {
        super.onResume();
        updateAutofillStatus();
        View rootView = this.getWindow().getDecorView();
```

(次のページに続く)

(前のページからの続き)

```
        if (!mIsAutofillEnabled) {
            // ホワイトリストに登録されている Autofill Service を利用しない場合は、
            // 全ての View を Autofill の対象外にする
            rootView.setImportantForAutofill(View.IMPORTANT_FOR_AUTOFILL_NO_EXCLUDE_
↪DESCENDANTS);
        } else {
            // ホワイトリストに登録されている Autofill Service を利用する場合は、
            // 全ての View を Autofill の対象にする
            // 特定の View に対して View#setImportantForAutofill() を呼び出すことも可能
            rootView.setImportantForAutofill(View.IMPORTANT_FOR_AUTOFILL_AUTO);
        }
    }
    private void login() {
        String username = mUsernameEditText.getText().toString();
        String password = mPasswordEditText.getText().toString();

        //View から取得したデータの安全性を確認する
        if (!Util.validateUsername(username) || !Util.validatePassword(password)) {
            //適切なエラー処理をする
        }

        //サーバーに username, password を送信

        finish();
    }

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

    private void updateAutofillStatus() {
        AutofillManager mgr = getSystemService(AutofillManager.class);
        // Android 9.0 (API Level 28) 以降では Autofill Service のコンポーネント情報を取得できるよ
うになった
        ComponentName componentName = mgr.getAutofillServiceComponentName();
        String componentNameString = "None";
        if (componentName == null) {
            mIsAutofillEnabled = false;//「設定」- 「自動入力サービス」が「なし」に設定されている
            Toast.makeText(this, "自動入力サービスなし", Toast.LENGTH_LONG).show();
        } else {
            String autofillServicePackage = componentName.getPackageName();
            // 使おうとしている Autofill Service がホワイトリストに登録されているかを検査する
            if (checkService(this, autofillServicePackage)) {
                mIsAutofillEnabled = true;
```

(次のページに続く)

(前のページからの続き)

```
                Toast.makeText(this, "信頼している自動入力サービス: " + autofillServicePackage,
                        Toast.LENGTH_LONG).show();
            } else {
                Toast.makeText(this, "信頼できない自動入力サービス: " + autofillServicePackage,
                        Toast.LENGTH_LONG).show();
                mIsAutofillEnabled = false;     //それ以外の場合は Autofill Service を利用しない
            }
            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;
```

(次のページに続く)

(前のページからの続き)

```
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 は 32 バイト
        sha256 = sha256.toUpperCase();
        if (sha256.replaceAll("[0-9A-F]+", "").length() != 0)
            return false;    // 0-9A-F 以外の文字がある

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

    public boolean test(Context ctx, String pkgname) {
        // pkgname に対応する正解のハッシュ値を取得する
        String correctHash = mWhitelists.get(pkgname);

        // pkgname の実際のハッシュ値と正解のハッシュ値を比較する
        if (Build.VERSION.SDK_INT >= 28) {
            // ★ API Level >= 28 では Package Manager の API で直接検証が可能
            PackageManager pm = ctx.getPackageManager();
            return pm.hasSigningCertificate(pkgname,
                                            Utils.hex2Bytes(correctHash),
                                            CERT_INPUT_SHA256);
        } else {
            // API Level < 28 の場合は PkgCert の機能を利用する
            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
 *
```

(次のページに続く)

(前のページからの続き)

```
 *      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);
            if (pkginfo.signatures.length != 1) return null;  // 複数署名は扱わない
            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) {
```

(次のページに続く)

(前のページからの続き)

```
        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 バックグラウンドアクティビティ起動（BAL）の制限強化

Android 17 では、ユーザーの意図しない不正な画面遷移や、マルウェアによる悪用を防ぐために、バックグラウンドアクティビティ起動（BAL: Background Activity Launch）に関する制限がさらに強化されている。従来の Android でも、アプリがバックグラウンド状態から直接画面（Activity）を起動することは一定条件下に制限されていたが、Android 17 ではその条件がより厳格になり、原則としてユーザーの明示的な操作がない状態での画面起動はできない方向に強化された。特に、通知やフォアグラウンドサービスなどを経由しても不自然なタイミングで画面を強制表示する挙動は制限対象となり、ユーザー体験の阻害やフィッシング的な不正誘導のリスクを抑える設計になっている。

この変更の背景には、バックグラウンドから突然ログイン画面や偽の入力画面を表示させて認証情報を盗み取るといった攻撃手法や、広告や不要な画面を強制的に表示する迷惑行為への対策がある。Android 17 では、こうした挙動を OS レベルで抑止することで、安全で予測可能なユーザー操作を重視する方向に進んでいる。そのため、開発者はこれまでのようにバックグラウンドから直接 Activity を起動するのではなく、通知（Notification）経由でユーザーにアクションを促し、タップなどの明示的操作をトリガーに画面を表示する設計へと移行する必要がある。このように BAL 制限の強化は、セキュリティと UX（ユーザー体験）の両方を守るための重要な変更となっている。

MODE_BACKGROUND_ACTIVITY_START_ALLOW_IF_VISIBLE は、Android におけるバックグラウンドアクティビティ起動（BAL）の制御の一つであり、アプリがユーザーに見えている状態にある場合に限って、バックグラウンドからの画面起動を許可するための設定である。Android では、バックグラウンドにあるアプリがユーザーの操作なしに突然画面を表示することは、フィッシングや不正な画面誘導、ユーザー体験の低下につながるため、基本的に厳しく制限されている。このモードを指定した場合、アプリがフォアグラウンドで動作している、もしくは画面上に一部でも表示されているといったユーザーが認識できる状態であれば、 PendingIntent などを通じて新しい Activity を起動することが可能になる。一方で、アプリが完全にバックグラウンドにあり、ユーザーから見えない状態では起動は許可されず、従来どおり通知を経由したユーザー操作が必要になる。このように可視かどうかを条件にすることで、ユーザーの現在の操作コンテキストを尊重しつつ、安全性を保つ設計になっている。

以下に、Android 17 で導入された MODE_BACKGROUND_ACTIVITY_START_ALLOW_IF_VISIBLE を指定して PendingIntent を送信する実装例を示す。

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

// ActivityOptions を生成
val options = ActivityOptions.makeBasic()

// 呼び出し元が可視状態のときのみバックグラウンドからの起動を許可する
options.pendingIntentBackgroundActivityStartMode =
    ActivityOptions.MODE_BACKGROUND_ACTIVITY_START_ALLOW_IF_VISIBLE

try {
    // オプションを Bundle に変換して PendingIntent を送信
    pendingIntent.send(
        context,
        0,
        intent,
        null,
        null,
        null,
        options.toBundle()
    )
} catch (e: PendingIntent.CanceledException) {
    e.printStackTrace()
}
```

4.2 Broadcast を受信する・送信する

4.2.1 サンプルコード

Broadcast を受信するには Broadcast Receiver を作る必要がある。どのような Broadcast を受信するかによって、Broadcast Receiver が抱えるリスクや適切な防御手段が異なる。次の判定フローによって作成する Broadcast Receiver がどのタイプであるかを判断できる。ちなみに、パートナー限定の連携に必要な Broadcast 送信元アプリのパッケージ名を受信側アプリで確認する手段がないため、パートナー限定 Broadcast Receiver を作る事はできない。

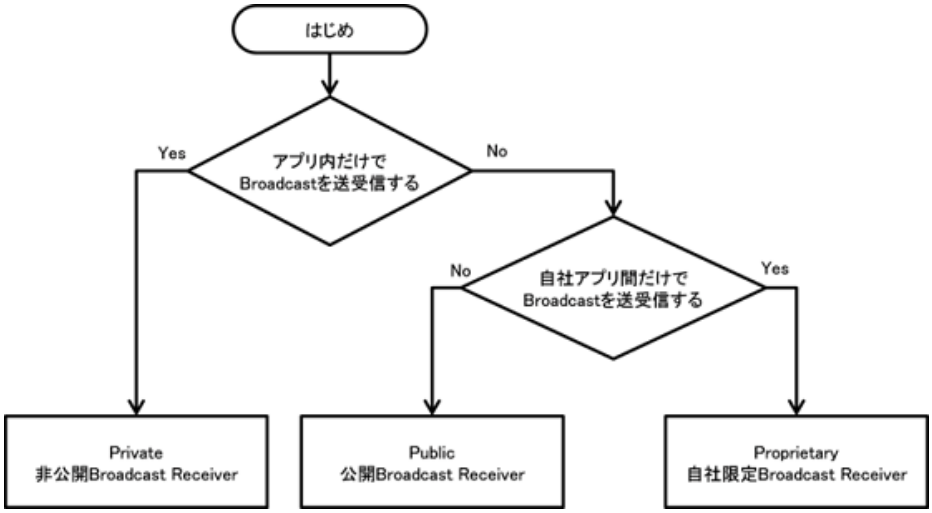

図 4.2.1　Broadcast Receiver のタイプ選択フロー

また Broadcast Receiver にはその定義方法により、静的 Broadcast Receiver と動的 Broadcast Receiver との 2 種類があり、下表のような特徴の違いがある。サンプルコード中では両方の実装方法を紹介している。なお、どのような相手に Broadcast を送信するかによって送信する情報の適切な防御手段が決まるため、送信側の実装についても合わせて説明する。

表 4.2.1　Broadcast Receiver の定義方法と特徴

	定義方法	特徴
静的 Broadcast Receiver	AndroidManifest.xml に<receiver>要素を記述することで定義する	• システムから送信される一部の Broadcast（ACTION_BATTERY_CHANGED など）を受信できない制約がある。 • アプリが初回起動してからアンインストールされるまでの間、Broadcast を受信できる。 • アプリの targetSDKVersion が 26 以上の場合、Android 8.0（API level 26) 以降の端末では、暗黙的 Broadcast Receiver を登録できない[10]
動的 Broadcast Receiver	プログラム中で registerReceiver() および unregisterReceiver() を呼び出すことにより、動的に Broadcast Receiver を登録／登録解除する	• 静的 Broadcast Receiver では受信できない Broadcast でも受信できる。 • Activity が前面に出ている期間だけ Broadcast を受信したいなど、Broadcast の受信可能期間をプログラムで制御できる。 • 非公開の Broadcast Receiver を作ることはできない。

4.2.1.1 非公開 Broadcast Receiver - Broadcast を受信する・送信する

非公開 Broadcast Receiver は、同一アプリ内から送信された Broadcast だけを受信できる Broadcast Receiver であり、もっとも安全性の高い Broadcast Receiver である。動的 Broadcast Receiver を非公開で登録することはできないため、非公開 Broadcast Receiver では静的 Broadcast Receiver だけで構成される。

[10] システムが送信する暗黙的 Broadcast Intent の中には例外的に利用可能なものも存在する。詳細は以下の URL を参照のこと https://developer.android.com/guide/components/broadcast-exceptions.html

ポイント (Broadcast を受信する) :

1. exported=”false” により、明示的に非公開設定する
2. 同一アプリ内から送信された Broadcast であっても、受信 Intent の安全性を確認する
3. 結果を返す場合、送信元は同一アプリ内であるから、センシティブな情報を返送してよい

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" >

    <!-- 非公開 Broadcast Receiver を定義する -->
    <!-- ★ポイント 1★ exported="false"により、明示的に非公開設定する -->
    <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
 *
```

(次のページに続く)

(前のページからの続き)

```
 * 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.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) {

        // ★ポイント 2★ 同一アプリ内から送信された Broadcast であっても、受信 Intent の安全性を確認する
        // サンプルにつき割愛。「3.2 入力データの安全性を確認する」を参照。
        String param = intent.getStringExtra("PARAM");
        Toast.makeText(context,
                       String.format("「%s」を受信した。", param),
                       Toast.LENGTH_SHORT).show();

        // ★ポイント 3★ 結果を返す場合、送信元は同一アプリ内であるから、センシティブな情報を返送してよい
        setResultCode(Activity.RESULT_OK);
        setResultData("センシティブな情報 from Receiver");
        abortBroadcast();
    }
}
```

次に非公開 Broadcast Receiver へ Broadcast 送信するサンプルコードを示す。

ポイント (Broadcast を送信する) :

4. 同一アプリ内 Receiver はクラス指定の明示的 Intent で Broadcast 送信する
5. 送信先は同一アプリ内 Receiver であるため、センシティブな情報を送信してよい
6. 同一アプリ内 Receiver からの結果情報であっても、受信データの安全性を確認する

PrivateSenderActivity.java

```
/*
 * Copyright (C) 2012-2026 Japan Smartphone Security Association
```

(次のページに続く)

(前のページからの続き)

```
 *
 * 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.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) {
        // ★ポイント 4★ 同一アプリ内 Receiver はクラス指定の明示的 Intent で Broadcast 送信する
        Intent intent = new Intent(this, PrivateReceiver.class);

        // ★ポイント 5★ 送信先は同一アプリ内 Receiver であるため、センシティブな情報を送信してよい
        intent.putExtra("PARAM", "センシティブな情報 from Sender");
        sendBroadcast(intent);
    }

    public void onSendOrderedClick(View view) {
        // ★ポイント 4★ 同一アプリ内 Receiver はクラス指定の明示的 Intent で Broadcast 送信する
        Intent intent = new Intent(this, PrivateReceiver.class);

        // ★ポイント 5★ 送信先は同一アプリ内 Receiver であるため、センシティブな情報を送信してよい
        intent.putExtra("PARAM", "センシティブな情報 from Sender");
        sendOrderedBroadcast(intent, null, mResultReceiver, null, 0, null, null);
    }

    private BroadcastReceiver mResultReceiver = new BroadcastReceiver() {
```

(次のページに続く)

(前のページからの続き)

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

                // ★ポイント 6★ 同一アプリ内 Receiver からの結果情報であっても、受信データの安全性を
確認する
                // サンプルにつき割愛。「3.2 入力データの安全性を確認する」を参照。
                String data = getResultData();
                PrivateSenderActivity.this.logLine(
                        String.format("結果「%s」を受信した。", 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");
    }
}
```

4.2.1.2 公開 Broadcast Receiver - Broadcast を受信する・送信する

公開 Broadcast Receiver は、不特定多数のアプリから送信された Broadcast を受信できる Broadcast Receiver である。マルウェアが送信した Broadcast を受信することがあることに注意が必要だ。

ポイント (Broadcast を受信する) :

1. exported=”true” により、明示的に公開設定する
2. 受信 Intent の安全性を確認する
3. 結果を返す場合、センシティブな情報を含めない

公開 Broadcast Receiver のサンプルコードである Public Receiver は、静的 Broadcast Receiver および動的 Broadcast Receiver の両方で利用される。

PublicReceiver.java

```
/*
 * Copyright (C) 2012-2026 Japan Smartphone Security Association
 *
```

(次のページに続く)

(前のページからの続き)

```
 * 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() {
        return isDynamic ? "公開動的 Broadcast Receiver" : "公開静的 Broadcast Receiver";
    }

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

        // ★ポイント 2★ 受信 Intent の安全性を確認する
        // 公開 Broadcast Receiver であるため利用元アプリがマルウェアである可能性がある。
        // サンプルにつき割愛。「3.2 入力データの安全性を確認する」を参照。
        if (MY_BROADCAST_PUBLIC.equals(intent.getAction())) {
            String param = intent.getStringExtra("PARAM");
            Toast.makeText(context,
                           String.format("%s:\n「%s」を受信した。", getName(), param),
                           Toast.LENGTH_SHORT).show();
        }

        // ★ポイント 3★ 結果を返す場合、センシティブな情報を含めない
```

(次のページに続く)

(前のページからの続き)

```
        // 公開 Broadcast Receiver であるため、
        // Broadcast の送信元アプリがマルウェアである可能性がある。
        // マルウェアに取得されても問題のない情報であれば結果として返してもよい。
        setResultCode(Activity.RESULT_OK);
        setResultData(String.format("センシティブではない情報 from %s", getName()));
        abortBroadcast();
    }
}
```

静的 Broadcast Receiver は AndroidManifest.xml で定義する。表 4.2.1 のように、端末のバージョンによって暗黙的 Broadcst Intent の受信に制限があるので注意すること。

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" >

    <!-- 公開静的 Broadcast Receiver を定義する -->
    <!-- ★ポイント 1★ exported="true"により、明示的に公開設定する -->
    <receiver android:name=".PublicReceiver"
              android:exported="true" >
      <intent-filter>
        <action android:name="org.jssec.android.broadcast.MY_BROADCAST_PUBLIC" />
      </intent-filter>
    </receiver>

    <service
        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>
```

(次のページに続く)

(前のページからの続き)

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

動的 Broadcast Receiver はプログラム中で registerReceiver() および unregisterReceiver() を呼び出すことにより登録／登録解除する。ボタン操作により登録／登録解除を行うために PublicReceiverActivity 上にボタンを配置している。動的 Broadcast Receiver インスタンスは PublicReceiverActivity より生存期間が長いため PublicReceiverActivity のメンバー変数として保持することはできない。そのため DynamicReceiverService のメンバー変数として動的 Broadcast Receiver のインスタンスを保持させ、DynamicReceiverService を PublicReceiverActivity から開始／終了することにより動的 Broadcast Receiver を間接的に登録／登録解除している。

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) {
        return null;
```

(次のページに続く)

(前のページからの続き)

```
    }

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

        // 公開動的 Broadcast Receiver を登録する
        mReceiver = new PublicReceiver();
        mReceiver.isDynamic = true;
        IntentFilter filter = new IntentFilter();
        filter.addAction(MY_BROADCAST_PUBLIC);
        filter.setPriority(1);  // 静的 Broadcast Receiver より動的 Broadcast Receiver を優先させる

        registerReceiver(mReceiver, filter);
        Toast.makeText(this,
                       "動的 Broadcast Receiver を登録した。",
                       Toast.LENGTH_SHORT).show();
    }

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

        // 公開動的 Broadcast Receiver を登録解除する
        unregisterReceiver(mReceiver);
        mReceiver = null;
        Toast.makeText(this,
                       "動的 Broadcast Receiver を登録解除した。",
                       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,
```

(次のページに続く)

(前のページからの続き)

```
 * 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;
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);
    }
}
```

次に公開 Broadcast Receiver へ Broadcast 送信するサンプルコードを示す。公開 Broadcast Receiver に Broadcast を送信する場合、送信する Broadcast がマルウェアに受信されることがあることに注意が必要である。

ポイント (Broadcast を送信する) :

4. センシティブな情報を送信してはならない

5. 結果を受け取る場合、結果データの安全性を確認する

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.
```

(次のページに続く)

(前のページからの続き)

```
 * 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 {

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

    public void onSendNormalClick(View view) {
        // ★ポイント 4★ センシティブな情報を送信してはならない
        Intent intent = new Intent(MY_BROADCAST_PUBLIC);
        intent.putExtra("PARAM", "センシティブではない情報 from Sender");
        sendBroadcast(intent);
    }

    public void onSendOrderedClick(View view) {
        // ★ポイント 4★ センシティブな情報を送信してはならない
        Intent intent = new Intent(MY_BROADCAST_PUBLIC);
        intent.putExtra("PARAM", "センシティブではない情報 from Sender");
        sendOrderedBroadcast(intent, null, mResultReceiver, null, 0, null, null);
    }

    public void onSendStickyClick(View view) {
        // ★ポイント 4★ センシティブな情報を送信してはならない
        Intent intent = new Intent(MY_BROADCAST_PUBLIC);
        intent.putExtra("PARAM", "センシティブではない情報 from Sender");
        //sendStickyBroadcast メソッドは API Level 21 で deprecated となった
```

(次のページに続く)

(前のページからの続き)

```
        sendStickyBroadcast(intent);
    }

    public void onSendStickyOrderedClick(View view) {
        // ★ポイント 4★ センシティブな情報を送信してはならない
        Intent intent = new Intent(MY_BROADCAST_PUBLIC);
        intent.putExtra("PARAM", "センシティブではない情報 from Sender");
        //sendStickyBroadcast メソッドは API Level 21 で deprecated となった
        sendStickyOrderedBroadcast(intent, mResultReceiver, null, 0, null, null);
    }

    public void onRemoveStickyClick(View view) {
        Intent intent = new Intent(MY_BROADCAST_PUBLIC);
        //removeStickyBroadcast メソッドは deprecated となっている
        removeStickyBroadcast(intent);
    }

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

            // ★ポイント 5★ 結果を受け取る場合、結果データの安全性を確認する
            // サンプルにつき割愛。「3.2 入力データの安全性を確認する」を参照。
            String data = getResultData();
            PublicSenderActivity.this.logLine(String.format("結果「%s」を受信した。",
                                                            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");
    }
}
```

4.2.1.3 自社限定 **Broadcast Receiver - Broadcast** を受信する・送信する

自社限定 Broadcast Receiver は、自社以外のアプリから送信された Broadcast を一切受信しない Broadcast Receiver である。複数の自社製アプリでシステムを構成し、自社アプリが扱う情報や機能を守るために利用される。

ポイント (Broadcast を受信する) :

1. Broadcast 受信用の独自定義 Signature Permission を定義する
2. 結果受信用の独自定義 Signature Permission を利用宣言する
3. exported=”true” により、明示的に公開設定する
4. 静的 Broadcast Receiver 定義にて、独自定義 Signature Permission を要求宣言する
5. 動的 Broadcast Receiver を登録するとき、独自定義 Signature Permission を要求宣言する
6. 独自定義 Signature Permission が自社アプリにより定義されていることを確認する
7. 自社アプリからの Broadcast であっても、受信 Intent の安全性を確認する
8. Broadcast 送信元は自社アプリであるから、センシティブな情報を返送してよい
9. APK を Export するときに、Broadcast 送信元アプリと同じ開発者鍵で APK を署名する

自社限定 Broadcast Receiver のサンプルコードである ProprietaryReciver は、静的 Broadcast Receiver および動的 Broadcast Receiver の両方で利用される。

InhouseReceiver.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.broadcast.inhousereceiver;

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

import android.app.Activity;
import android.content.BroadcastReceiver;
```

(次のページに続く)

(前のページからの続き)

```
import android.content.Context;
import android.content.Intent;
import android.widget.Toast;

public class InhouseReceiver extends BroadcastReceiver {

    // 自社の Signature Permission
    private static final String MY_PERMISSION =
        "org.jssec.android.broadcast.inhousereceiver.MY_PERMISSION";

    // 自社の証明書のハッシュ値
    private static String sMyCertHash = null;
    private static String myCertHash(Context context) {
        if (sMyCertHash == null) {
            if (Utils.isDebuggable(context)) {
                // debug.keystore の"androiddebugkey"の証明書ハッシュ値
                sMyCertHash = "0EFB7236 328348A9 89718BAD DF57F544 D5CCB4AE B9DB34BC␣
↪1E29DD26 F77C8255";
            } else {
                // keystore の"my company key"の証明書ハッシュ値
                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 ? "自社限定動的 Broadcast Receiver" : "自社限定静的 Broadcast Receiver
↪";
    }

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

        // ★ポイント 6★ 独自定義 Signature Permission が自社アプリにより定義されていることを確認する
        if (!SigPerm.test(context, MY_PERMISSION, myCertHash(context))) {
            Toast.makeText(context,
                           "独自定義 Signature Permission が自社アプリにより定義されていない。",
                           Toast.LENGTH_LONG).show();
            return;
```

(次のページに続く)

(前のページからの続き)

```
        }

        // ★ポイント 7★ 自社アプリからの Broadcast であっても、受信 Intent の安全性を確認する
        // サンプルにつき割愛。「3.2 入力データの安全性を確認する」を参照。
        if (MY_BROADCAST_INHOUSE.equals(intent.getAction())) {
            String param = intent.getStringExtra("PARAM");
            Toast.makeText(context,
                           String.format("%s:\n「%s」を受信した。", getName(), param),
                           Toast.LENGTH_SHORT).show();
        }

        // ★ポイント 8★ Broadcast 送信元は自社アプリであるから、センシティブな情報を返送してよい
        setResultCode(Activity.RESULT_OK);
        setResultData(String.format("センシティブな情報 from %s", getName()));
        abortBroadcast();
    }
}
```

静的 Broadcast Receiver は AndroidManifest.xml で定義する。表 4.2.1 のように、端末のバージョンによって暗黙的 Broadcst Intent の受信に制限があるので注意すること。

AndroidManifest.xml

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

  <!-- ★ポイント 1★ Broadcast 受信用の独自定義 Signature Permission を定義する -->
  <permission
      android:name="org.jssec.android.broadcast.inhousereceiver.MY_PERMISSION"
      android:protectionLevel="signature" />

  <!-- ★ポイント 2★ 結果受信用の独自定義 Signature Permission を利用宣言する -->
  <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">

    <!-- ★ポイント 3★ exported="true"により、明示的に公開設定する -->
    <!-- ★ポイント 4★ 静的 Broadcast Receiver 定義にて、独自定義 Signature Permission を要求宣言する -->
    <receiver
        android:name="org.jssec.android.broadcast.inhousereceiver.InhouseReceiver"
```

(次のページに続く)

(前のページからの続き)

```
        android:exported="true"
        android:permission="org.jssec.android.broadcast.inhousereceiver.MY_PERMISSION">
      <intent-filter>
        <action android:name="org.jssec.android.broadcast.MY_BROADCAST_INHOUSE" />
      </intent-filter>
    </receiver>

    <service
        android:name="org.jssec.android.broadcast.inhousereceiver.DynamicReceiverService"
        android:exported="false" />

    <activity
        android:name="org.jssec.android.broadcast.inhousereceiver.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>
```

動的 Broadcast Receiver はプログラム中で registerReceiver() および unregisterReceiver() を呼び出すことにより登録／登録解除する。ボタン操作により登録／登録解除を行うために ProprietaryReceiverActivity 上にボタンを配置している。動的 Broadcast Receiver インスタンスは ProprietaryReceiverActivity より生存期間が長いため ProprietaryReceiverActivity のメンバー変数として保持することはできない。そのため DynamicReceiverService のメンバー変数として動的 Broadcast Receiver のインスタンスを保持させ、DynamicReceiverService を ProprietaryReceiverActivity から開始／終了することにより動的 Broadcast Receiver を間接的に登録／登録解除している。

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
```

(次のページに続く)

(前のページからの続き)

```
 * 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);
    }

    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.
 */
```

(次のページに続く)

(前のページからの続き)

```
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);
        filter.setPriority(1);  // 静的 Broadcast Receiver より動的 Broadcast Receiver を優先させる

        // ★ポイント 5★ 動的 Broadcast Receiver を登録するとき、独自定義 Signature Permission を要求宣言する
        registerReceiver(mReceiver, filter,
                "org.jssec.android.broadcast.inhousereceiver.MY_PERMISSION",
                null);

        Toast.makeText(this,
                       "動的 Broadcast Receiver を登録した。",
                       Toast.LENGTH_SHORT).show();
    }

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

(次のページに続く)

(前のページからの続き)

```
        unregisterReceiver(mReceiver);
        mReceiver = null;
        Toast.makeText(this,
                       "動的 Broadcast Receiver を登録解除した。",
                       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(" ", "");
        try{
            // sigPermName を定義したアプリのパッケージ名を取得する
            PackageManager pm = ctx.getPackageManager();
            PermissionInfo pi =
```

(次のページに続く)

(前のページからの続き)

```
                pm.getPermissionInfo(sigPermName, PackageManager.GET_META_DATA);
            String pkgname = pi.packageName;
            // 非 Signature Permission の場合は失敗扱い
            if (pi.protectionLevel != PermissionInfo.PROTECTION_SIGNATURE) return false;
            // pkgname の実際のハッシュ値と正解のハッシュ値を比較する
            if (Build.VERSION.SDK_INT >= 28) {
                // ★ API Level >= 28 では Package Manager の API で直接検証が可能
                return pm.hasSigningCertificate(pkgname,
                                                Utils.hex2Bytes(correctHash),
                                                CERT_INPUT_SHA256);
            } else {
                // API Level < 28 の場合は PkgCert を利用し、ハッシュ値を取得して比較する
                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;
```

(次のページに続く)

(前のページからの続き)

```
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);
            if (pkginfo.signatures.length != 1) return null;  // 複数署名は扱わない
            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();
    }
}
```

★ポイント 9 ★ APK を Export するときに、Broadcast 送信元アプリと同じ開発者鍵で APK を署名する。

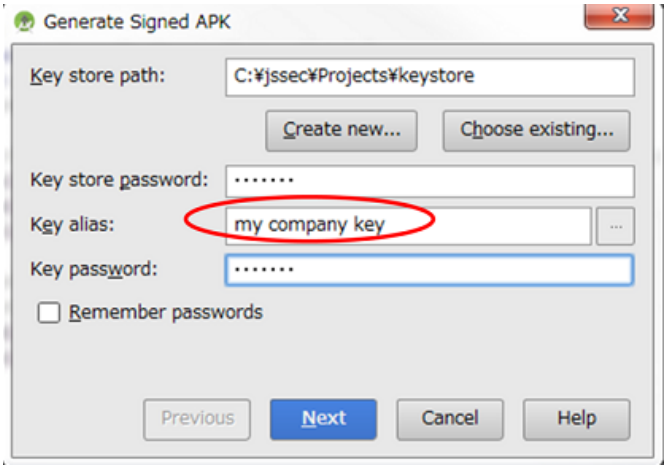

図 4.2.2　Broadcast 送信元アプリと同じ開発者鍵で APK を署名する

次に自社限定 Broadcast Receiver へ Broadcast 送信するサンプルコードを示す。

ポイント (Broadcast を送信する) :

10. 結果受信用の独自定義 Signature Permission を定義する
11. Broadcast 受信用の独自定義 Signature Permission を利用宣言する
12. 独自定義 Signature Permission が自社アプリにより定義されていることを確認する
13. Receiver は自社アプリ限定であるから、センシティブな情報を送信してもよい
14. Receiver に独自定義 Signature Permission を要求する
15. 結果を受け取る場合、結果データの安全性を確認する
16. APK を Export するときに、Receiver 側アプリと同じ開発者鍵で APK を署名する

```
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"/>

  <!-- ★ポイント 10★ 結果受信用の独自定義 Signature Permission を定義する -->
  <permission
      android:name="org.jssec.android.broadcast.inhousesender.MY_PERMISSION"
      android:protectionLevel="signature" />

  <!-- ★ポイント 11★ Broadcast 受信用の独自定義 Signature Permission を利用宣言する -->
```

(次のページに続く)

(前のページからの続き)

```
  <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
        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;
```

(次のページに続く)

(前のページからの続き)

```
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 {

    // 自社の Signature Permission
    private static final String MY_PERMISSION =
        "org.jssec.android.broadcast.inhousesender.MY_PERMISSION";

    // 自社の証明書のハッシュ値
    private static String sMyCertHash = null;
    private static String myCertHash(Context context) {
        if (sMyCertHash == null) {
            if (Utils.isDebuggable(context)) {
                // debug.keystore の"androiddebugkey"の証明書ハッシュ値
                sMyCertHash = "0EFB7236 328348A9 89718BAD DF57F544 D5CCB4AE B9DB34BC␣
↪1E29DD26 F77C8255";
            } else {
                // keystore の"my company key"の証明書ハッシュ値
                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) {

        // ★ポイント 12★ 独自定義 Signature Permission が自社アプリにより定義されていることを確認す
る
        if (!SigPerm.test(this, MY_PERMISSION, myCertHash(this))) {
            Toast.makeText(this,
                           "独自定義 Signature Permission が自社アプリにより定義されていない。",
                           Toast.LENGTH_LONG).show();
            return;
        }

        // ★ポイント 13★ Receiver は自社アプリ限定であるから、センシティブな情報を送信してもよい
        Intent intent = new Intent(MY_BROADCAST_INHOUSE);
```

(次のページに続く)

(前のページからの続き)

```
        intent.putExtra("PARAM", "センシティブな情報 from Sender");

        // ★ポイント 14★ Receiver に独自定義 Signature Permission を要求する
        sendBroadcast(intent, "org.jssec.android.broadcast.inhousesender.MY_PERMISSION");
    }

    public void onSendOrderedClick(View view) {

        // ★ポイント 12★ 独自定義 Signature Permission が自社アプリにより定義されていることを確認す
る
        if (!SigPerm.test(this, MY_PERMISSION, myCertHash(this))) {
            Toast.makeText(this,
                           "独自定義 Signature Permission が自社アプリにより定義されていない。",
                            Toast.LENGTH_LONG).show();
            return;
        }

        // ★ポイント 13★ Receiver は自社アプリ限定であるから、センシティブな情報を送信してもよい
        Intent intent = new Intent(MY_BROADCAST_INHOUSE);
        intent.putExtra("PARAM", "センシティブな情報 from Sender");

        // ★ポイント 14★ Receiver に独自定義 Signature Permission を要求する
        sendOrderedBroadcast(intent,
                             "org.jssec.android.broadcast.inhousesender.MY_PERMISSION",
                              mResultReceiver, null, 0, null, null);
    }

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

            // ★ポイント 15★ 結果を受け取る場合、結果データの安全性を確認する
            // サンプルにつき割愛。「3.2 入力データの安全性を確認する」を参照。
            String data = getResultData();
            InhouseSenderActivity.this.logLine(String.format("結果「%s」を受信した。", data));
        }
    };

    private TextView mLogView;

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

(次のページに続く)

(前のページからの続き)

```
    }

    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;

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{
            // sigPermName を定義したアプリのパッケージ名を取得する
            PackageManager pm = ctx.getPackageManager();
            PermissionInfo pi =
```

(次のページに続く)

(前のページからの続き)

```
                pm.getPermissionInfo(sigPermName, PackageManager.GET_META_DATA);
            String pkgname = pi.packageName;
            // 非 Signature Permission の場合は失敗扱い
            if (pi.protectionLevel != PermissionInfo.PROTECTION_SIGNATURE) return false;
            // pkgname の実際のハッシュ値と正解のハッシュ値を比較する
            if (Build.VERSION.SDK_INT >= 28) {
                // ★ API Level >= 28 では Package Manager の API で直接検証が可能
                return pm.hasSigningCertificate(pkgname,
                                                Utils.hex2Bytes(correctHash),
                                                CERT_INPUT_SHA256);
            } else {
                // API Level < 28 の場合は PkgCert を利用し、ハッシュ値を取得して比較する
                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;
```

(次のページに続く)

(前のページからの続き)

```
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);
            if (pkginfo.signatures.length != 1) return null;  // 複数署名は扱わない
            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();
    }
}
```

★ポイント 16 ★ APK を Export するときに、Receiver 側アプリと同じ開発者鍵で APK を署名する。

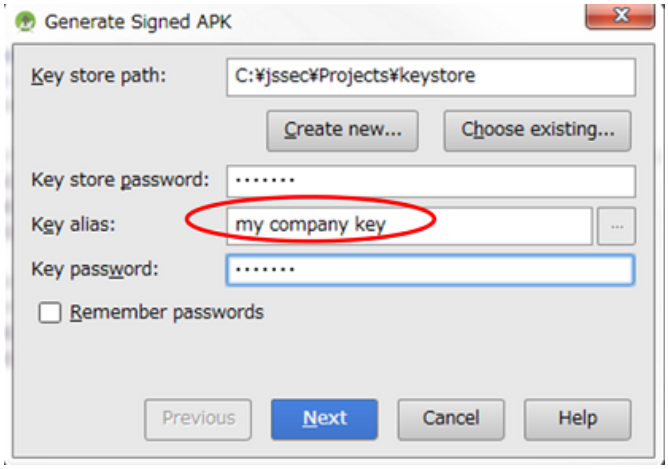

図 4.2.3　Reciver 側アプリと同じ開発者鍵で APK を署名する

4.2.2 ルールブック

Broadcast を送受信する際には以下のルールを守ること。

1. アプリ内でのみ使用する *Broadcast Receiver* は非公開設定する（必須）
2. 受信 *Intent* の安全性を確認する（必須）
3. 独自定義 *Signature Permission* は、自社アプリが定義したことを確認して利用する（必須）
4. 結果情報を返す場合には、返送先アプリからの結果情報漏洩に注意する（必須）
5. センシティブな情報を *Broadcast* 送信する場合は、受信可能な *Receiver* を制限する（必須）
6. *Sticky Broadcast* にはセンシティブな情報を含めない（必須）
7. *receiverPermission* パラメータの指定なし *Ordered Broadcast* は届かないことがあることに注意（必須）
8. *Broadcast Receiver* からの返信データの安全性を確認する（必須）
9. 資産を二次的に提供する場合には、その資産の従来の保護水準を維持する（必須）

4.2.2.1 アプリ内でのみ使用する **Broadcast Receiver** は非公開設定する（必須）

アプリ内でのみ使用される Broadcast Receiver は非公開設定する。これにより、他のアプリから意図せず Broadcast を受け取ってしまうことがなくなり、アプリの機能を利用されたり、アプリの動作に異常をきたしたりするのを防ぐことができる。

同一アプリ内からのみ利用される Receiver では Intent Filter を設置するような設計はしてはならない。Intent Filter の性質上、同一アプリ内の非公開 Receiver を呼び出すつもりでも、Intent Filter 経由で呼び出したときに意図せず他アプリの公開 Receiver を呼び出してしまう場合が存在するからである。

AndroidManifest.xml(非推奨)

```
    <!-- 外部アプリに非公開とする Broadcast Receiver -->
    <!-- 4.2.1.1 - ポイント 1: exported="false"により、明示的に非公開設定する -->
```

(次のページに続く)

（前のページからの続き）

```
<receiver android:name=".PrivateReceiver"
    android:exported="false" >
    <intent-filter>
        <action android:name="org.jssec.android.broadcast.MY_ACTION" />
    </intent-filter>
</receiver>
```

「4.2.3.1. *exported* 設定と *intent-filter* 設定の組み合わせ *(Receiver* の場合*)*」も参照すること。

4.2.2.2 受信 Intent の安全性を確認する（必須）

Broadcast Receiver のタイプによって若干リスクは異なるが、受信 Intent のデータを処理する際には、まず受信 Intent の安全性を確認しなければならない。

公開 Broadcast Receiver は不特定多数のアプリから Intent を受け取るため、マルウェアの攻撃 Intent を受け取る可能性がある。非公開 Broadcast Receiver は他のアプリから Intent を直接受け取ることはない。しかし同一アプリ内の公開 Component が他のアプリから受け取った Intent のデータを非公開 Broadcast Receiver に転送することがあるため、受信 Intent を無条件に安全であると考えてはならない。自社限定 Broadcast Receiver はその中間のリスクであるため、やはり受信 Intent の安全性を確認する必要がある。

「3.2. 入力データの安全性を確認する」を参照すること。

4.2.2.3 独自定義 Signature Permission は、自社アプリが定義したことを確認して利用する（必須）

自社のアプリから送信された Broadcast だけを受信し、それ以外の Broadcast を一切受信しない自社限定 Broadcast Receiver を作る場合、独自定義 Signature Permission により保護しなければならない。AndroidManifest.xml での Permission 定義、Permission 要求宣言だけでは保護が不十分であるため、「5.2. *Permission* と *Protection Level*」の「5.2.1.2. 独自定義の *Signature Permission* で自社アプリ連携する方法」を参照すること。また独自定義 Signature Permission を receiverPermission パラメータに指定して Broadcast 送信する場合も同様に確認する必要がある。

4.2.2.4 結果情報を返す場合には、返送先アプリからの結果情報漏洩に注意する（必須）

Broadcast Receiver のタイプによって setResult() により結果情報を返すアプリの信用度が異なる。公開 Broadcast Receiver の場合は、結果返送先のアプリがマルウェアである可能性もあり、結果情報が悪意を持って使われる危険性がある。非公開 Broadcast Receiver や自社限定 Broadcast Receiver の場合は、結果返送先は自社開発アプリであるため結果情報の扱いをあまり心配する必要はない。

このように Broadcast Receiver から結果情報を返す場合には、返送先アプリからの結果情報の漏洩に配慮しなければならない。

4.2.2.5 センシティブな情報を Broadcast 送信する場合は、受信可能な Receiver を制限する（必須）

Broadcast という名前が表すように、そもそも Broadcast は不特定多数のアプリに情報を一斉送信したり、タイミングを通知したりすることを意図して作られた仕組みである。そのためセンシティブな情報を Broadcast 送信する場合には、マルウェアに情報を取得されないような注意深い設計が必要となる。

センシティブな情報を Broadcast 送信する場合、信頼できる Broadcast Receiver だけが受信可能であり、他の Broadcast Receiver は受信不可能である必要がある。そのための Broadcast 送信方法には以下のようなものがある。

- 明示的 Intent で Broadcast 送信することで宛先を固定し、意図した信頼できる Broadcast Receiver だけに Broadcast を届ける方法。この方法には次の 2 つのパターンがある。
  - 同一アプリ内 Broadcast Receiver 宛てであれば Intent#setClass(Context, Class) により宛先を限定する。具体的なコードについてはサンプルコードセクション「4.2.1.1. 非公開 *Broadcast Receiver - Broadcast* を受信する・送信する」を参考にすること。
  - 他のアプリの Broadcast Receiver 宛てであれば Intent#setClassName(String, String) により宛先を限定するが、Broadcast 送信に先立ち宛先パッケージの APK 署名の開発者鍵をホワイトリストと照合して許可したアプリであることを確認してから Broadcast を送信する。実際には暗黙的 Intent を利用できる次の方法が実用的である。
- receiverPermission パラメータに独自定義 Signature Permission を指定して Broadcast 送信し、信頼する Broadcast Receiver に当該 Signature Permission を利用宣言してもらう方法。具体的なコードについてはサンプルコードセクション「4.2.1.3. 自社限定 *Broadcast Receiver - Broadcast* を受信する・送信する」を参考にすること。またこの Broadcast 送信方法を実装するにはルール「4.2.2.3. 独自定義 *Signature Permission* は、自社アプリが定義したことを確認して利用する（必須）」も適用しなければならない。

4.2.2.6 Sticky Broadcast にはセンシティブな情報を含めない（必須）

通常の Broadcast は、Broadcast 送信時に受信可能状態にある Broadcast Receiver に受信処理されると、その Broadcast は消滅してしまう。一方 Sticky Broadcast（および Sticky Ordered Broadcast、以下同様）は、送信時に受信状態にある Broadcast Receiver に受信処理された後もシステム上に存在しつづけ、その後 registerReceiver() により受信できることが特徴である。不要になった Sticky Broadcast は removeStickyBroadcast() により任意のタイミングで削除できる。

Sticky Broadcast は暗黙的 Intent による使用が前提であり、receiverPermission パラメータを指定した Broadcast 送信はできない。そのため Sticky Broadcast で送信した情報はマルウェアを含む不特定多数のアプリから取得できてしまう。したがってセンシティブな情報を Sticky Broadcast で送信してはならない。なお、Sticky Broadcast の使用は Android 5.0（API Level 21）において非推奨となっている。

4.2.2.7 receiverPermission パラメータの指定なし Ordered Broadcast は届かないことがあることに注意（必須）

receiverPermission パラメータを指定せずに送信された Ordered Broadcast は、マルウェアを含む不特定多数のアプリが受信可能である。Ordered Broadcast は Receiver からの返り情報を受け取るため、または複数の Receiver に順次処理をさせるために利用される。優先度の高い Receiver から順次 Broadcast が配送されるため、優先度を高くしたマルウェアが最初に Broadcast を受信し abortBroadcast() すると、後続の Receiver に Broadcast が配信されなくなる。

4.2.2.8 Broadcast Receiver からの返信データの安全性を確認する（必須）

結果データを送り返してきた Broadcast Receiver のタイプによって若干リスクは異なるが、基本的には受信した結果データが攻撃データである可能性を考慮して安全に処理しなければならない。

返信元 Broadcast Receiver が公開 Broadcast Receiver の場合、不特定のアプリから戻りデータを受け取るため、マルウェアの攻撃データを受け取る可能性がある。返信元 Broadcast Receiver が非公開 Broadcast Receiver の場合、同一アプリ内からの結果データであるのでリスクはないように考えがちだが、他のアプリから受け取ったデータを間接的に結果データとして転送することがあるため、結果データを無条件に安全であると考えてはならない。返信元 Broadcast Receiver が

自社限定 Broadcast Receiver の場合、その中間のリスクであるため、やはり結果データが攻撃データである可能性を考慮して安全に処理しなければならない。

「3.2. 入力データの安全性を確認する」を参照すること。

4.2.2.9 資産を二次的に提供する場合には、その資産の従来の保護水準を維持する（必須）

Permission により保護されている情報資産および機能資産を他のアプリに二次的に提供する場合には、提供先アプリに対して同一の Permission を要求するなどして、その保護水準を維持しなければならない。Android の Permission セキュリティモデルでは、保護された資産に対するアプリからの直接アクセスについてのみ権限管理を行う。この仕様上の特性により、アプリに取得された資産がさらに他のアプリに、保護のために必要な Permission を要求することなく提供される可能性がある。このことは Permission を再委譲していることと実質的に等価なので、Permission の再委譲問題と呼ばれる。「5.2.3.4. *Permission* の再委譲問題」を参照すること。

4.2.3 アドバンスト

4.2.3.1 exported 設定と intent-filter 設定の組み合わせ (Receiver の場合)

「表 4.2.2」は、Receiver を実装するときに使用可能な exported 属性と intent-filter 要素の組み合わせを示している。

表 4.2.2　exported 設定と intent-filter 設定の組み合わせ

exported 属性の値	intent-filter 定義がある	intent-filter 定義がない
true	可	可
false	原則禁止	可
無指定	禁止	禁止

本ガイドでは、Receiver の exported 属性を「無指定」にすることを一律で禁止している。OS や intent-filter の有無による API のデフォルト挙動に依存した実装は避けるべきであり[*11]、exported 属性のようなセキュリティ上重要な設定は、常に明示的に指定して有効化すべきであるからである。また、Android 12（API Level 31）以降をターゲットとするアプリにおいて、intent-filter を定義しながら exported 属性を無指定にした場合は、ビルドエラー（マニフェストマージエラー）となるため物理的にも動作しない。

「exported=”false” かつ intent-filter 定義あり」の組み合わせを原則禁止としているのは、同一アプリ内の非公開 Receiver に向けて Broadcast を送信したつもりでも、意図せず他アプリの公開 Receiver を呼び出してしまう場合が存在するからである。以下の 2 つの図で意図せぬ呼び出しが起こる様子を説明する。

図 4.2.4 は、同一アプリ内からしか非公開 Receiver(アプリ A）を暗黙的 Intent で呼び出せない正常な動作の例である。Intent-filter(図中 action=”X”) を定義しているのが、アプリ A しかいないので意図通りの動きとなっている。

[*11] intent-filter が定義されていれば公開 Receiver、定義されていなければ非公開 Receiver となる。https://developer.android.com/guide/topics/manifest/receiver-element.html#exported を参照のこと。

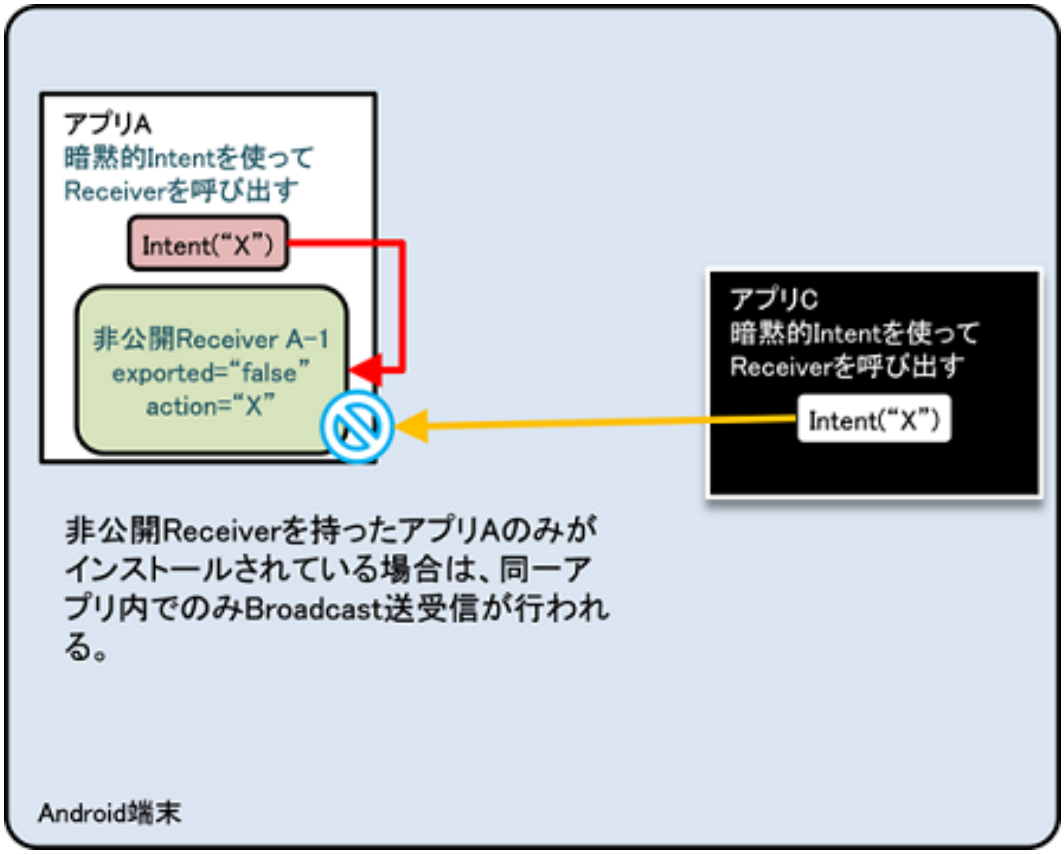

図 4.2.4　正常な動作の例

図 4.2.5 は、アプリ A に加えてアプリ B でも同じ intent-filter(図中 action=”X”) を定義している場合である。まず、他のアプリ (アプリ C) が暗黙的 Intent で Broadcast を送信するのは、非公開 Receiver(A-1) 側は受信をしないので特にセキュリティ的には問題にならない (図の橙色の矢印)。

セキュリティ面で問題になるのは、アプリ A による同一アプリ内の非公開 Receiver の呼び出しである。アプリ A が暗黙的 Intent を Broadcast すると、同一アプリ内の非公開 Receiver に加えて、同じ intent-filter を定義したアプリ B の持つ公開 Receiver（B-1）もその Intent を受信できてしまう（図の赤色の矢印)。これにより、アプリ A からアプリ B に対してセンシティブな情報を送信してしまう可能性が生じる。アプリ B がマルウェアであれば、そのままセンシティブな情報の漏洩に繋がる。また、Broadcast が Ordered Broadcast であった場合は、意図しない結果情報を受け取ってしまう可能性もある。

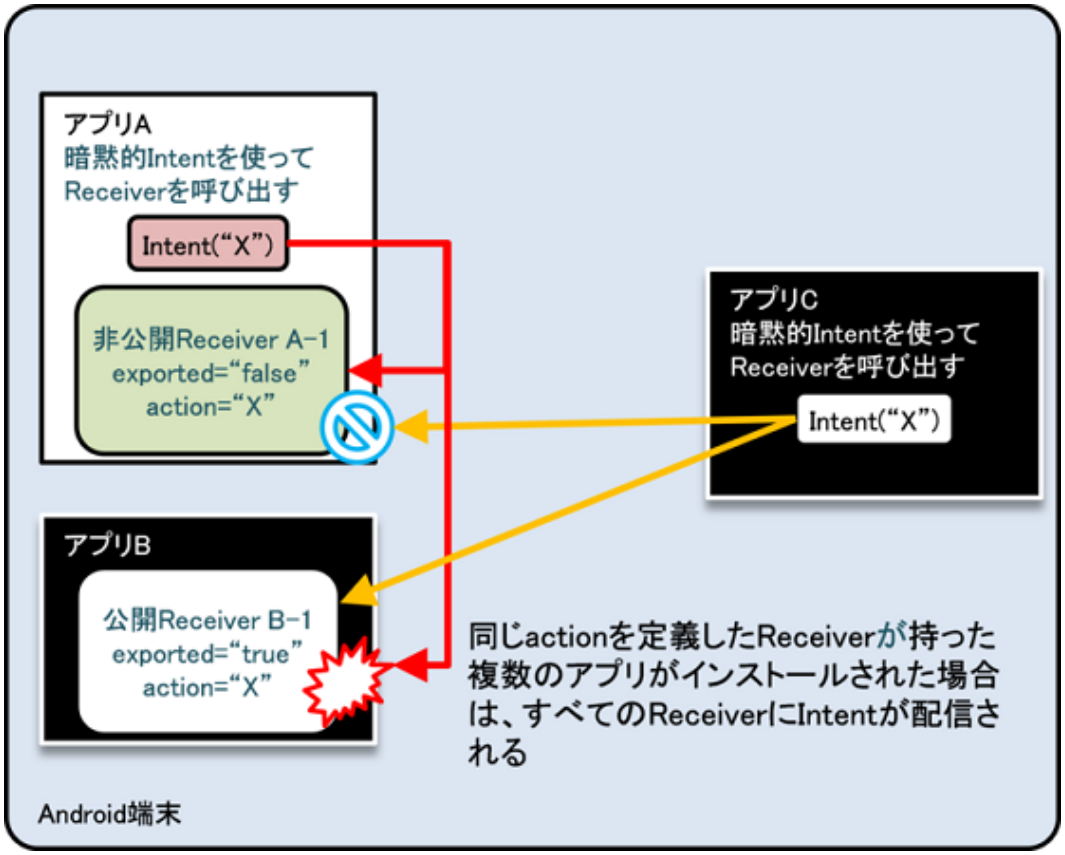

図 4.2.5　意図しない動作の例

ただし、システムが送信する Broadcast Intent のみを受信する BroadcastReceiver を実装する場合には、この「exported=”false” かつ intent-filter 定義あり」を使用しなければならない（かつ、これ以外の組み合わせは使用してはならない）。これは、システムが送信する Broadcast Intent に関しては、exported=”false” であっても安全に受信可能であるという OS の仕様に基づく。システムが送信する Broadcast Intent と同じ ACTION の Intent を他アプリが送信した場合、それを受信してしまうと意図しない動作を引き起こす可能性があるが、これは exported=”false” を指定することによって完全に防ぐことができるためである。

なお、Android 12 以降、exported 設定は必須となっている。詳細は、4.7.3.1. コンポーネント公開制御と *Intent* 送信の制限 を参照すること。

4.2.3.2 Receiver はアプリを起動しないと登録されない

AndroidManifest.xml に静的に定義した Broadcast Receiver は、インストールしただけでは有効にならないので注意が必要である[*12]。アプリを 1 回起動することで、それ以降の Broadcast を受信できるようになるため、インストール後に Broadcast 受信をトリガーにして処理を起動させることはできない。ただし Broadcast の送信側で Intent に Intent.FLAG_INCLUDE_STOPPED_PACKAGES を設定して Broadcast 送信した場合は、一度も起動していないアプリであってもこの Broadcast を受信することができる。

[*12] Android 3.0 未満ではアプリのインストールをしただけで Receiver が登録される

4.2.3.3 同じ **UID** を持つアプリから送信された **Broadcast** は、非公開 **Broadcast Receiver** でも受信できる

複数のアプリに同じ UID を持たせることができる。この場合、たとえ非公開 Broadcast Receiver であっても、同じ UID のアプリから送信された Broadcast は受信してしまう。

しかしこれはセキュリティ上問題となることはない。同じ UID を持つアプリは APK を署名する開発者鍵が一致することが保証されており、非公開 Broadcast Receiver が受信するのは自社アプリから送信された Broadcast に限定されるからである。

4.2.3.4 **Broadcast** の種類とその特徴

送信する Broadcast は Ordered かそうでないか、Sticky かそうでないかの組み合わせにより 4 種類の Broadcast が存在する。Broadcast 送信用メソッドに応じて、送信する Broadcast の種類が決まる。なお、Sticky Broadcast の使用は Android 5.0（API Level 21）において非推奨となっている。

表 4.2.3　送信する Broadcast の種類

Broadcast の種類	送信用メソッド	**Ordered?**	**Sticky?**
Normal Broadcast	sendBroadcast()	No	No
Ordered Broadcast	sendOrderedBroadcast()	Yes	No
Sticky Broadcast	sendStickyBroadcast()	No	Yes
Sticky Ordered Broadcast	sendStickyOrderedBroadcast()	Yes	Yes

それぞれの Broadcast の特徴は次のとおりである。

表 4.2.4　送信する Broadcast の種類ごとの特徴

Broadcast の種類	**Broadcast** の種類ごとの特徴
Normal Broadcast	Normal Broadcast は送信時に受信可能な状態にある Broadcast Receiver に配送されて消滅する。Ordered Broadcast と異なり、複数の Broadcast Receiver が同時に Broadcast を受信するのが特徴である。特定の Permission を持つアプリの BroadcastReceiver だけに Broadcast を受信させることもできる。
Ordered Broadcast	Ordered Broadcast は送信時に受信可能な状態にある BroadcastReceiver が一つずつ順番に Broadcast を受信することが特徴である。より priority 値が大きい Broadcast Receiver が先に受信する。すべての Broadcast Receiver に配送完了するか、途中の Broadcast Receiver が abortBroadcast() を呼び出した場合に、Broadcast は消滅する。特定の Permission を利用宣言したアプリの Broadcast Receiver だけに Broadcast を受信させることもできる。また Ordered Broadcast では送信元が Broadcast Receiver からの結果情報を受け取ることもできる。SMS 受信通知 Broadcast（SMS_RECEIVED）は Ordered Broadcast の代表例である。
Sticky Broadcast	Sticky Broadcast は送信時に受信可能な状態にある BroadcastReceiver に配送された後に消滅することはなくシステムに残り続け、後に registerReceiver() を呼び出したアプリが Sticky Broadcast を受信することができることが特徴である。Sticky Broadcast は他の Broadcast と異なり自動的に消滅することはないので、Sticky Broadcast が不要になったときに、明示的に removeStickyBroadcast() を呼び出して Sticky Broadcast を消滅させる必要がある。他の Broadcast と異なり、特定の Permission を持つアプリの Broadcast Receiver だけに Broadcast を受信させることはできない。バッテリー状態変更通知 Broadcast （ACTION_BATTERY_CHANGED）は Sticky Broadcast の代表例である。
Sticky Ordered Broadcast	Ordered Broadcast と Sticky Broadcast の両方の特徴を持った Broadcast である。Sticky Broadcast と同様、特定の Permission を持つアプリの Broadcast Receiver だけに Broadcast を受信させることはできない。

Broadcast の特徴的な振る舞いの視点で、上表を逆引き的に再整理すると下表になる。

表 4.2.5　Broadcast の特徴的な振る舞い

Broadcast の特徴的な振る舞い	**Normal Broadcast**	**Ordered Broadcast**	**Sticky Broadcast**	**Sticky Ordered Broadcast**
受信可能な Broadcast Receive を Permission により制限する	o	o	-	-
Broadcast Receiver からの処理結果を取得する	-	o	-	o
順番に Broadcast Receiver に Broadcast を処理させる	-	o	-	o
既に送信されている Broadcast を後から受信する	-	-	o	o

4.2.3.5 Broadcast 送信した情報が LogCat に出力される場合がある

Broadcast の送受信は基本的に LogCat に出力されない。しかし、受信側の Permission 不足によるエラーや、送信側の Permission 不足によるエラーの際に LogCat にエラーログが出力される。エラーログには Broadcast で送信する Intent 情報も含まれるので、エラー発生時には Broadcast 送信する場合は LogCat に表示されることに注意してほしい。

送信側の Permission 不足時のエラー

```
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
```

受信側の Permission 不足時のエラー

```
W/ActivityManager(275): Permission Denial: receiving Intent {
act=org.jssec.android.broadcastreceiver.creating.action.MY_ACTION } to
org.jssec.android.broadcastreceiver.creating requires
org.jssec.android.permission.MY_PERMISSION due to sender
org.jssec.android.broadcast.sending (uid 10158)
```

4.2.3.6 ホーム画面（アプリ）にショートカットを配置する際の注意点

ホーム画面にアプリを起動するためのショートカットボタンや Web ブラウザのブックマークのような URL ショートカットを作成する場合の注意点について説明する。例として、以下のような実装を考えてみる。

ホーム画面（アプリ）にショートカットを配置する

```
Intent targetIntent = new Intent(this, TargetActivity.class);

// ショートカット作成依頼のための Intent
Intent intent = new Intent("com.android.launcher.action.INSTALL_SHORTCUT");

// ショートカットのタップ時に起動する Intent を指定
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);

// BroadCast を使って、システムにショートカット作成を依頼する
context.sendBroadcast(intent);
```

上記で送信している Broadcast は、受け手がホーム画面アプリであり、パッケージ名を特定することが難しいため、暗黙的 Intent による公開 Receiver への送信となっていることに注意が必要である。つまり、上記で送信した Broadcast はマルウェアを含めた任意のアプリが受信することができ、そのため、Intent にセンシティブな情報が含まれていると情報漏

洩の被害につながる可能性がある。URL を基にしたショートカットを作成する場合、URL に秘密の情報が含まれていることもあるため、特に注意が必要である。

対策方法としては、「4.2.1.2. 公開 *Broadcast Receiver - Broadcast* を受信する・送信する」に記載されているポイントに従い、送信する Intent にセンシティブな情報が含まないようにすることが必要である。

4.2.3.7 ACTION_CLOSE_SYSTEM_DIALOGS について

ACTION_CLOSE_SYSTEM_DIALOGS は、システムダイアログが閉じられたことを意味する Broadcast Intent である。また、アプリから Broadcast 送信することで、システムダイアログを閉じることができる。この ACTION_CLOSE_SYSTEM_DIALOGS は、Android 12 をターゲットとするか Android11 以前をターゲットとするかで以下の通り挙動が変更になっている。

Android 11 以前をターゲットとする場合

インテントは実行されず logcat には次のメッセージが表示される。

```
E ActivityTaskManager Permission Denial: \
android.intent.action.CLOSE_SYSTEM_DIALOGS broadcast from \
com.package.name requires android.permission.BROADCAST_CLOSE_SYSTEM_DIALOGS, \
dropping broadcast.
```

ただし、次の場合においては、引き続きシステムダイアログを閉じることができる。

- 通知ドロワーの上に Window を表示している
- ユーザーが通知を操作し、アプリがそのユーザーアクションに応じてサービスまたは Broadcast Receiver を処理している
- ユーザー補助サービスが有効になっている

Android 12 をターゲットとする場合

ACTION_CLOSE_SYSTEM_DIALOGS は非推奨となった。アプリがこのアクションを含むインテントの呼び出しを試みると SecurityException が発生する。

ただし、次の場合においては、引き続きシステムダイアログを閉じることができる。

- アプリがインストゥルメント化単体テストを実行している場合

ユーザー補助サービスが有効になっており、通知バーを閉じる必要がある場合は、代わりに GLOBAL_ACTION_DISMISS_NOTIFICATION_SHADE ユーザー補助アクションを使用する。

4.2.3.8 動的 Broadcast Receiver の安全性の強化

これまで動的 Broadcast Receiver は export 設定が不可であったが、Android 13 からは公開/非公開を指定できるようになった。これは、マルウェアが送信した Broadcast を受信することがあるリスクを考慮しての、安全性の強化の観点からなされた仕様である。

この機能を利用するには targetSdkVersion を 33 以上に設定し以下のフローで行う。

1. DYNAMIC_RECEIVER_EXPLICIT_EXPORT_REQUIRED を有効にする
2. registerReceiver の引数に RECEIVER_EXPORTED あるいは RECEIVER_NOT_EXPORTED を指定する

```
context.registerReceiver(sharedBroadcastReceiver, intentFilter, RECEIVER_EXPORTED);
context.registerReceiver(sharedBroadcastReceiver, intentFilter, RECEIVER_NOT_EXPORTED);
```

DYNAMIC_RECEIVER_EXPLICIT_EXPORT_REQUIRED の設定は開発者オプションもしくは adb から切り替え可能である。開発者オプションから設定する場合、以下のような画面から行う。

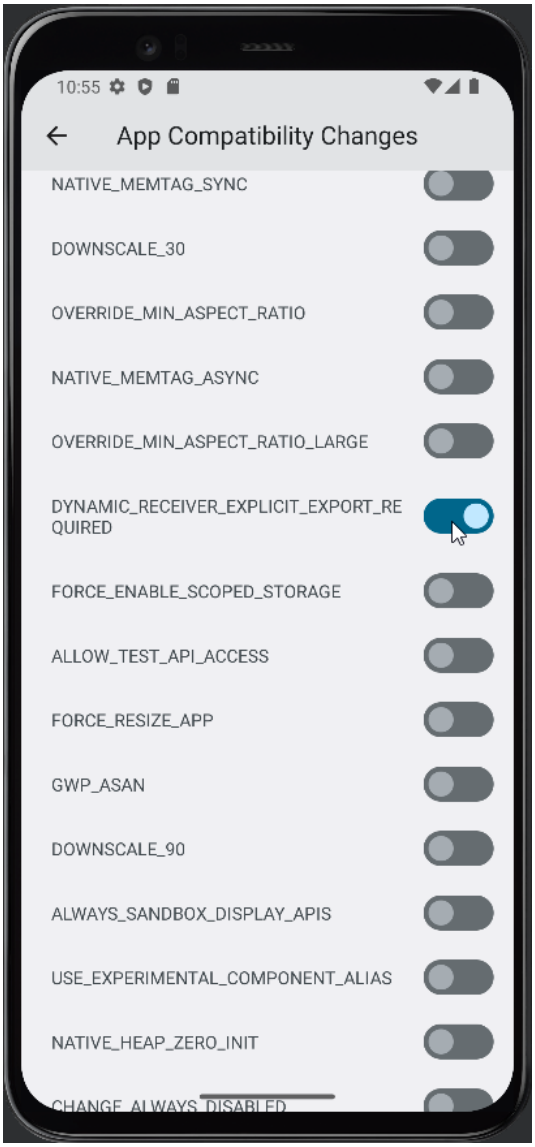

図 4.2.6　DYNAMIC_RECEIVER_EXPLICIT_EXPORT_REQUIRED の設定 (開発者オプション)

また、adb から設定する場合は、以下のようなコマンドラインとなる。

```
$ 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
```

DYNAMIC_RECEIVER_EXPLICIT_EXPORT_REQUIRED を有効にした場合、RECEIVER_EXPORTED あるいは RECEIVER_NOT_EXPORTED を指定せず registerReceiver() を実行すると、SecurityException が発生する。

DYNAMIC_RECEIVER_EXPLICIT_EXPORT_REQUIRED を無効にした場合、RECEIVER_EXPORTED あるいは RE-

CEIVER_NOT_EXPORTED を指定して registerReceiver() を実行すると、export 設定は無視され、従来通り公開 Broadcast Receiver として動作する。

なお、Android 14 からは、DYNAMIC_RECEIVER_EXPLICIT_EXPORT_REQUIRED はデフォルトで有効となっており、したがって、Android 14 をターゲットとしたアプリにおいては export 設定の指定は必須となっている。

ただし、ACTION_TIME_TICK などシステム Broadcast[*13] のみを受信する Receiver の場合、export 設定は不要である。たとえ設定しても執筆時点においてはビルド時、実行時ともエラーや警告などは発生しないが、将来的な仕様への追従を考慮するなら、このような Receiver においては export 設定しないことが推奨される。

- システム Broadcast である ACTION_TIME_TICK を受信する Broadcast Receiver の登録例

```
IntentFilter filter = new IntentFilter(Intent.ACTION_TIME_TICK);
receiver = new MyBroadcastReceiver();
registerReceiver(receiver, filter);
```

4.2.3.9 順序付き Broadcast の priority 仕様変更

Android 16 から、順序付き Broadcast の priority 指定に関する仕様が大きく変わった。これまでの Android では、IntentFilter#setPriority() や android:priority 属性を使うことで、複数の BroadcastReceiver 間で受信順序を細かく制御できた。たとえば priority 値を高く設定した Receiver を「必ず最初に処理したい」などの要件がある場合、priority 値に依存した設計が一般的だった。

しかし、Android 16 以降は priority 値の扱いが根本から変更された。priority 値をいくら高くしても、システムは 0 として扱うため、priority 指定による順序制御は事実上できなくなった。アプリ内で複数の Receiver を登録している場合も、priority 値ではなく登録順やシステムの都合によって呼び出し順が決まる。これにより、priority 値に依存した順序制御は完全に崩れることになる。

たとえば、次のようなコードは Android 15 以前なら priority=1000 の ReceiverA が必ず先に呼ばれていたが、Android 16 以降は順序が保証されない。

```
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)
filterB.priority = 0
```

(次のページに続く)

[*13] 一覧は API Level によって異なり 34 の場合は以下に一覧の記載がある。/AppData/Local/Android/Sdk/platforms/android-34/data/broadcast_actions.txt

(前のページからの続き)

```
registerReceiver(receiverA, filterA, RECEIVER_NOT_EXPORTED)
registerReceiver(receiverB, filterB, RECEIVER_NOT_EXPORTED)

sendOrderedBroadcast(Intent(ACTION_TEST).apply { `package` = packageName }, null)
```

Android 16 以降、このコードは ReceiverA → ReceiverB の順で呼ばれる保証がなくなり、場合によっては ReceiverB が先に呼ばれることもある。priority 値に依存した設計は、今後の OS アップデートや Receiver の追加・削除によって予期せぬバグやセキュリティ事故につながるリスクが高い。

この仕様変更を受けて、開発者は priority 値に依存した順序制御をやめ、確実に順序を保証したい処理は BroadcastReceiver で分割せず、1 つの関数やクラスで明示的に順序を担保する設計に切り替える必要がある。たとえば、A → B の順序で処理したい場合は次のように書き換える。

```
private fun doOrderedProcess() {
    stepA()
    stepB()
}

private fun stepA() {
    Log.d("OrderedProcess", "A の処理")
    // A の処理内容
}

private fun stepB() {
    Log.d("OrderedProcess", "B の処理")
    // B の処理内容
}
```

このように関数呼び出しで順序を明示すれば、Android のバージョンやシステムの挙動に左右されず、必ず意図した順序で処理できる。もし Receiver を分離したい場合は、最初の Receiver で必要な処理をすべて順番に呼び出すなど、順序を自分で担保する設計に見直すことが必要になる。

Broadcast は本来、複数の独立したコンポーネントやアプリに同じイベントを通知したい場合に使うものであり、順序保証が必須の処理には向かない。順序が必要な場合は Broadcast や priority に頼らず、明示的な呼び出しや Service、コールバックなどで自前で順序を担保する設計に切り替えることが、Android 16 以降の安全な開発の基本となる。

4.2.3.10 Bluetooth ボンドの損失に対する自律的な再ペア設定

Android 17 以降では、Bluetooth 接続の切断（ボンドの損失）を自動的に解決するためのシステムレベルの拡張機能である「自律的な再ペア設定」が導入された。

従来は、周辺機器とのバインドが失われた場合、ユーザー自身が設定画面に手動で移動し、ペア設定を解除してから再度ペア設定を行う必要があった。Android 17 では Android 16 のセキュリティ強化を基盤として、ユーザーが手動で設定に移動して周辺機器のペア設定を解除し再ペア設定する必要なく、システムがバックグラウンドで接続を再確立できるようになる。

ほとんどのアプリではコードの変更は必要ないが、スマートウォッチや IoT デバイスなど、周辺機器と連携して動作す

る専用アプリや Bluetooth 通信を行うアプリの開発者は、Bluetooth スタックにおける以下の動作変更について認識しておく必要がある。

- インテントのタイミングの変更: ACTION_KEY_MISSING インテントは、自律的な再ペア設定の試行が失敗した場合にのみブロードキャストされるようになった。これにより、システムがバックグラウンドでボンドを正常に復元した場合に、アプリで不要なエラー処理を行う必要がなくなる。
- 新しいペア設定コンテキスト: ACTION_PAIRING_REQUEST インテントに EXTRA_PAIRING_CONTEXT エクストラが含まれるようになった。これにより、アプリは標準のペア設定リクエストと、システムが自律的に開始した再ペア設定の試行を区別できる。
- 条件付き鍵の更新: 再ペア設定が成功し、新しい接続が以前のボンドのセキュリティレベルを満たすか、それを超える場合にのみ、既存のセキュリティ鍵が置き換えられる。これはセキュリティレベルのダウングレード攻撃を防ぐ保護として機能する。
- ユーザー通知: システムは、新しい UI 通知とダイアログを介して再ペア設定を管理する。ユーザーには、再ペア設定の試行を確認するよう求めるメッセージが表示され、再接続を認識できるようになっている。

周辺機器メーカーや連携アプリの開発者は、ハードウェアとアプリがバインドの移行を適切に処理することを確認する必要がある。この動作をテストするには、周辺機器から手動でペア設定情報を削除するか、Android の「設定 > 接続済みのデバイス」から手動でペア設定を解除することで、リモートボンドの損失をシミュレートすることが推奨される。

4.3 Content Provider を作る・利用する

ContentResolver と SQLiteDatabase のインターフェースが似ているため、Content Provider は SQLiteDatabase と密接に関係があると勘違いされることが多い。しかし実際には Content Provider はアプリ間データ共有のインターフェースを規定するだけで、データ保存の形式は一切問わないことに注意が必要だ。作成する Content Provider 内部でデータの保存に SQLiteDatabase を使うこともできるし、XML ファイルなどの別の保存形式を使うこともできる。なお、ここで紹介するサンプルコードにはデータを保存する処理を含まないので、必要に応じて追加すること。

4.3.1 サンプルコード

Content Provider がどのように利用されるかによって、Content Provider が抱えるリスクや適切な防御手段が異なる。次の判定フローによって作成する Content Provider がどのタイプであるかを判断できる。

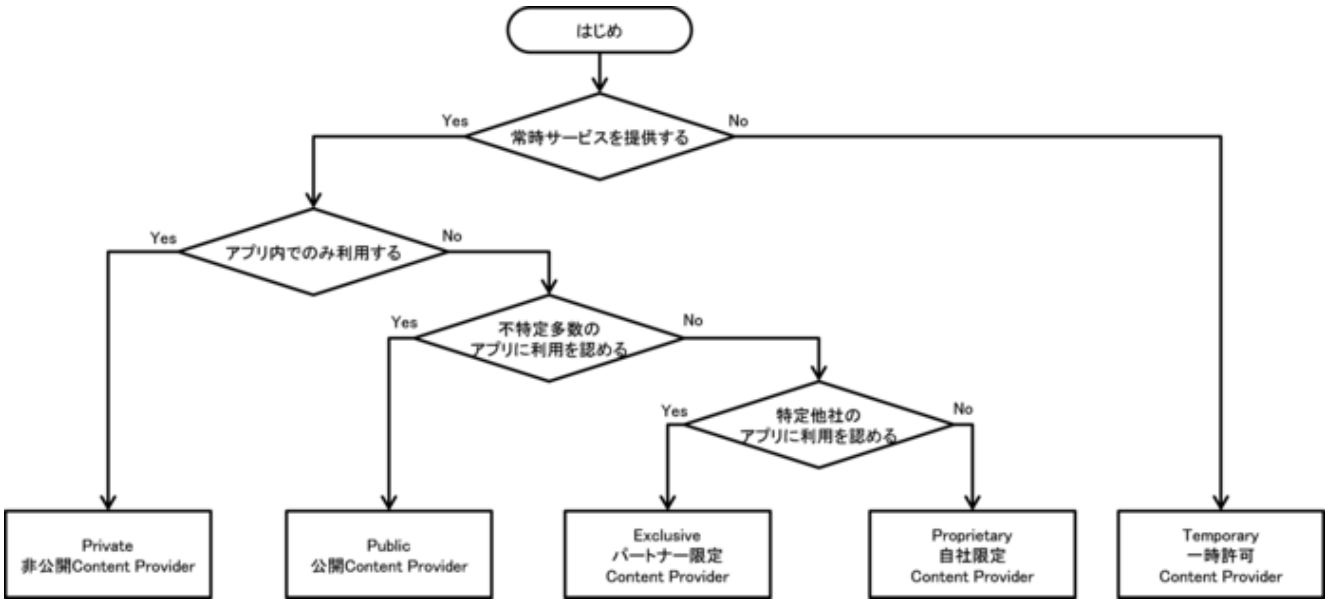

図 4.3.1　Content Provider タイプ判定フロー

4.3.1.1 非公開 Content Provider を作る・利用する

非公開 Content Provider は、同一アプリ内だけで利用される Content Provider であり、もっとも安全性の高い Content Provider である[*14]。

以下、非公開 Content Provider の実装例を示す。

ポイント (Content Provider を作る)：

1. exported=”false” により、明示的に非公開設定する
2. 同一アプリ内からのリクエストであっても、パラメータの安全性を確認する
3. 利用元アプリは同一アプリであるから、センシティブな情報を返送してよい

```
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" />
```

(次のページに続く)

[*14] ただし、Content Provider の非公開設定は Android 2.2 (API Level 8) 以前では機能しない。

(前のページからの続き)

```
      </intent-filter>
    </activity>

    <!-- ★ポイント 1★ exported="false"により、明示的に非公開設定する -->
    <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;
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 =
```

(次のページに続く)

(前のページからの続き)

```
        "vnd.android.cursor.item/vnd.org.jssec.contenttype";

    // Content Provider が提供するインターフェースを公開
    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);
    }

    // DB を使用せずに固定値を返す例にしているため、query メソッドで返す Cursor を事前に定義
    private static MatrixCursor sAddressCursor =
        new MatrixCursor(new String[] { "_id", "pref" });
    static {
        sAddressCursor.addRow(new String[] { "1", "北海道" });
        sAddressCursor.addRow(new String[] { "2", "青森" });
        sAddressCursor.addRow(new String[] { "3", "岩手" });
    }
    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() {
```

(次のページに続く)

(前のページからの続き)

```
        return true;
    }

    @Override
    public String getType(Uri uri) {

        // ★ポイント 2★ 同一アプリ内からのリクエストであっても、パラメータの安全性を確認する
        // ここでは uri が想定の範囲内であることを、UriMatcher#match() と switch case で確認している。
        // 「3.2 入力データの安全性を確認する」を参照。
        // ★ポイント 3★ 利用元アプリは同一アプリであるから、センシティブな情報を返送してよい
        // ただし getType の結果がセンシティブな意味を持つことはあまりない。
        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) {

        // ★ポイント 2★ 同一アプリ内からのリクエストであっても、パラメータの安全性を確認する
        // ここでは uri が想定の範囲内であることを、UriMatcher#match() と switch case で確認している。
        // その他のパラメータの確認はサンプルにつき省略。「3.2 入力データの安全性を確認する」を参照。
        // ★ポイント 3★ 利用元アプリは同一アプリであるから、センシティブな情報を返送してよい
        // query の結果がセンシティブな意味を持つかどうかはアプリ次第。
        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);
```

(次のページに続く)

(前のページからの続き)

```
    }
}

@Override
public Uri insert(Uri uri, ContentValues values) {

    // ★ポイント 2★ 同一アプリ内からのリクエストであっても、パラメータの安全性を確認する
    // ここでは uri が想定の範囲内であることを、UriMatcher#match() と switch case で確認している。
    // その他のパラメータの確認はサンプルにつき省略。「3.2 入力データの安全性を確認する」を参照。
    // ★ポイント 3★ 利用元アプリは同一アプリであるから、センシティブな情報を返送してよい
    // Insert 結果、発番される ID がセンシティブな意味を持つかどうかはアプリ次第。
    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) {

    // ★ポイント 2★ 同一アプリ内からのリクエストであっても、パラメータの安全性を確認する
    // ここでは uri が想定の範囲内であることを、UriMatcher#match() と switch case で確認している。
    // その他のパラメータの確認はサンプルにつき省略。「3.2 入力データの安全性を確認する」を参照。
    // ★ポイント 3★ 利用元アプリは同一アプリであるから、センシティブな情報を返送してよい
    // Update されたレコード数がセンシティブな意味を持つかどうかはアプリ次第。
    switch (sUriMatcher.match(uri)) {
    case DOWNLOADS_CODE:
        return 5;       // update されたレコード数を返す

    case DOWNLOADS_ID_CODE:
        return 1;

    case ADDRESSES_CODE:
        return 15;

    case ADDRESSES_ID_CODE:
        return 1;
```

(次のページに続く)

(前のページからの続き)

```
        default:
            throw new IllegalArgumentException("Invalid URI：" + uri);
        }
    }

    @Override
    public int delete(Uri uri, String selection, String[] selectionArgs) {

        // ★ポイント 2★ 同一アプリ内からのリクエストであっても、パラメータの安全性を確認する
        // ここでは uri が想定の範囲内であることを、UriMatcher#match() と switch case で確認している。
        // その他のパラメータの確認はサンプルにつき省略。「3.2 入力データの安全性を確認する」を参照。
        // ★ポイント 3★ 利用元アプリは同一アプリであるから、センシティブな情報を返送してよい
        // Delete されたレコード数がセンシティブな意味を持つかどうかはアプリ次第。
        switch (sUriMatcher.match(uri)) {
        case DOWNLOADS_CODE:
            return 10;      // delete されたレコード数を返す

        case DOWNLOADS_ID_CODE:
            return 1;

        case ADDRESSES_CODE:
            return 20;

        case ADDRESSES_ID_CODE:
            return 1;

        default:
            throw new IllegalArgumentException("Invalid URI：" + uri);
        }
    }
}
```

次に、非公開 Content Provider を利用する Activity の例を示す。

ポイント (Content Provider を利用する)：

4. 同一アプリ内へのリクエストであるから、センシティブな情報をリクエストに含めてよい
5. 同一アプリ内からの結果情報であっても、受信データの安全性を確認する

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.
```

(次のページに続く)

(前のページからの続き)

```
 * 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.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]");

        // ★ポイント 4★ 同一アプリ内へのリクエストであるから、センシティブな情報をリクエストに含めてよい
        Cursor cursor = null;
        try {
            cursor = getContentResolver().query(PrivateProvider.Download.CONTENT_URI,
                                                null, null, null, null);

            // ★ポイント 5★ 同一アプリ内からの結果情報であっても、受信データの安全性を確認する
            // サンプルにつき割愛。「3.2 入力データの安全性を確認する」を参照。
            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();
                }
```

(次のページに続く)

(前のページからの続き)

```
            }
        }
        finally {
            if (cursor != null) cursor.close();
        }
    }

    public void onInsertClick(View view) {

        logLine("[Insert]");

        // ★ポイント 4★ 同一アプリ内へのリクエストであるから、センシティブな情報をリクエストに含めてよ
い
        Uri uri = getContentResolver().insert(PrivateProvider.Download.CONTENT_URI, null);

        // ★ポイント 5★ 同一アプリ内からの結果情報であっても、受信データの安全性を確認する
        // サンプルにつき割愛。「3.2 入力データの安全性を確認する」を参照。
        logLine("  uri:" + uri);
    }

    public void onUpdateClick(View view) {

        logLine("[Update]");

        // ★ポイント 4★ 同一アプリ内へのリクエストであるから、センシティブな情報をリクエストに含めてよ
い
        int count =
            getContentResolver().update(PrivateProvider.Download.CONTENT_URI, null, null,␣
↪null);

        // ★ポイント 5★ 同一アプリ内からの結果情報であっても、受信データの安全性を確認する
        // サンプルにつき割愛。「3.2 入力データの安全性を確認する」を参照。
        logLine(String.format("  %s records updated", count));
    }

    public void onDeleteClick(View view) {

        logLine("[Delete]");

        // ★ポイント 4★ 同一アプリ内へのリクエストであるから、センシティブな情報をリクエストに含めてよ
い
        int count =
            getContentResolver().delete(PrivateProvider.Download.CONTENT_URI, null, null);

        // ★ポイント 5★ 同一アプリ内からの結果情報であっても、受信データの安全性を確認する
```

(次のページに続く)

(前のページからの続き)

```
        // サンプルにつき割愛。「3.2 入力データの安全性を確認する」を参照。
        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");
    }
}
```

4.3.1.2 公開 Content Provider を作る・利用する

公開 Content Provider は、不特定多数のアプリに利用されることを想定した Content Provider である。クライアントを限定しないことにより、マルウェアから select() によって保持しているデータを抜き取られたり、update() によってデータを書き換えられたり、insert()/delete() によって偽のデータの挿入やデータの削除といった攻撃を受けたりして改ざんされ得ることに注意が必要だ。

また、Android OS 既定ではない独自作成の公開 Content Provider を利用する場合、その公開 Content Provider に成り済ましたマルウェアにリクエストパラメータを受信されることがあること、および、攻撃結果データを受け取ることがあることに注意が必要である。Android OS 既定の Contacts や MediaStore 等も公開 Content Provider であるが、マルウェアはそれら Content Provider に成り済ましできない。

以下、公開 Content Provider の実装例を示す。

ポイント (Content Provider を作る) :

1. exported=”true” により、明示的に公開設定する
2. リクエストパラメータの安全性を確認する
3. センシティブな情報を返送してはならない

AndroidManifest.xml

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

  <application
```

(次のページに続く)

(前のページからの続き)

```
      android:icon="@drawable/ic_launcher"
      android:label="@string/app_name" >

    <!-- ★ポイント 1★ exported="true"により、明示的に公開設定する -->
    <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
 * 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 =
```

(次のページに続く)

(前のページからの続き)

```
        "vnd.android.cursor.item/vnd.org.jssec.contenttype";

    // Content Provider が提供するインターフェースを公開
    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);
    }

    // DB を使用せずに固定値を返す例にしているため、query メソッドで返す Cursor を事前に定義
    private static MatrixCursor sAddressCursor =
        new MatrixCursor(new String[] { "_id", "pref" });
    static {
        sAddressCursor.addRow(new String[] { "1", "北海道" });
        sAddressCursor.addRow(new String[] { "2", "青森" });
        sAddressCursor.addRow(new String[] { "3", "岩手" });
    }
    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() {
```

(次のページに続く)

(前のページからの続き)

```
        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) {

        // ★ポイント 2★ リクエストパラメータの安全性を確認する
        // ここでは uri が想定の範囲内であることを、UriMatcher#match() と switch case で確認している。
        // その他のパラメータの確認はサンプルにつき省略。「3.2 入力データの安全性を確認する」を参照。

        // ★ポイント 3★ センシティブな情報を返送してはならない
        // query の結果がセンシティブな意味を持つかどうかはアプリ次第。
        // リクエスト元のアプリがマルウェアである可能性がある。
        // マルウェアに取得されても問題のない情報であれば結果として返してもよい。
        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);
        }
    }
```

(次のページに続く)

(前のページからの続き)

```
@Override
public Uri insert(Uri uri, ContentValues values) {

    // ★ポイント 2★ リクエストパラメータの安全性を確認する
    // ここでは uri が想定の範囲内であることを、UriMatcher#match() と switch case で確認している。
    // その他のパラメータの確認はサンプルにつき省略。「3.2 入力データの安全性を確認する」を参照。

    // ★ポイント 3★ センシティブな情報を返送してはならない
    // Insert 結果、発番される ID がセンシティブな意味を持つかどうかはアプリ次第。
    // リクエスト元のアプリがマルウェアである可能性がある。
    // マルウェアに取得されても問題のない情報であれば結果として返してもよい。
    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) {

    // ★ポイント 2★ リクエストパラメータの安全性を確認する
    // ここでは uri が想定の範囲内であることを、UriMatcher#match() と switch case で確認している。
    // その他のパラメータの確認はサンプルにつき省略。「3.2 入力データの安全性を確認する」を参照。

    // ★ポイント 3★ センシティブな情報を返送してはならない
    // Update されたレコード数がセンシティブな意味を持つかどうかはアプリ次第。
    // リクエスト元のアプリがマルウェアである可能性がある。
    // マルウェアに取得されても問題のない情報であれば結果として返してもよい。
    switch (sUriMatcher.match(uri)) {
    case DOWNLOADS_CODE:
        return 5;       // update されたレコード数を返す

    case DOWNLOADS_ID_CODE:
        return 1;

    case ADDRESSES_CODE:
        return 15;
```

(次のページに続く)

(前のページからの続き)

```
        case ADDRESSES_ID_CODE:
            return 1;

        default:
            throw new IllegalArgumentException("Invalid URI:" + uri);
        }
    }

    @Override
    public int delete(Uri uri, String selection, String[] selectionArgs) {

        // ★ポイント 2★ リクエストパラメータの安全性を確認する
        // ここでは uri が想定の範囲内であることを、UriMatcher#match() と switch case で確認している。
        // その他のパラメータの確認はサンプルにつき省略。「3.2 入力データの安全性を確認する」を参照。

        // ★ポイント 3★ センシティブな情報を返送してはならない
        // Delete されたレコード数がセンシティブな意味を持つかどうかはアプリ次第。
        // リクエスト元のアプリがマルウェアである可能性がある。
        // マルウェアに取得されても問題のない情報であれば結果として返してもよい。
        switch (sUriMatcher.match(uri)) {
        case DOWNLOADS_CODE:
            return 10;      // delete されたレコード数を返す

        case DOWNLOADS_ID_CODE:
            return 1;

        case ADDRESSES_CODE:
            return 20;

        case ADDRESSES_ID_CODE:
            return 1;

        default:
            throw new IllegalArgumentException("Invalid URI:" + uri);
        }
    }
}
```

次に、公開 Content Provider を利用する Activity の例を示す。

ポイント (Content Provider を利用する) :

4. センシティブな情報をリクエストに含めてはならない
5. 結果データの安全性を確認する

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.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 {

    // 利用先の Content Provider 情報
    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 が不在");
            return;
```

(次のページに続く)

(前のページからの続き)

```
        }

        // ★ポイント 4★ センシティブな情報をリクエストに含めてはならない
        // リクエスト先のアプリがマルウェアである可能性がある。
        // マルウェアに取得されても問題のない情報であればリクエストに含めてもよい。
        Cursor cursor = null;
        try {
            cursor =
                getContentResolver().query(Address.CONTENT_URI, null, null, null, null);

            // ★ポイント 5★ 結果データの安全性を確認する
            // サンプルにつき割愛。「3.2 入力データの安全性を確認する」を参照。
            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 が不在");
            return;
        }

        // ★ポイント 4★ センシティブな情報をリクエストに含めてはならない
        // リクエスト先のアプリがマルウェアである可能性がある。
        // マルウェアに取得されても問題のない情報であればリクエストに含めてもよい。
        ContentValues values = new ContentValues();
        values.put("pref", "東京都");
        Uri uri = getContentResolver().insert(Address.CONTENT_URI, values);
```

(次のページに続く)

(前のページからの続き)

```
        // ★ポイント 5 ★ 結果データの安全性を確認する
        // サンプルにつき割愛。「3.2 入力データの安全性を確認する」を参照。
        logLine("  uri:" + uri);
    }

    public void onUpdateClick(View view) {

        logLine("[Update]");

        if (!providerExists(Address.CONTENT_URI)) {
            logLine("  Content Provider が不在");
            return;
        }

        // ★ポイント 4 ★ センシティブな情報をリクエストに含めてはならない
        // リクエスト先のアプリがマルウェアである可能性がある。
        // マルウェアに取得されても問題のない情報であればリクエストに含めてもよい。
        ContentValues values = new ContentValues();
        values.put("pref", "東京都");
        String where = "_id = ?";
        String[] args = { "4" };
        int count = getContentResolver().update(Address.CONTENT_URI, values, where, args);

        // ★ポイント 5 ★ 結果データの安全性を確認する
        // サンプルにつき割愛。「3.2 入力データの安全性を確認する」を参照。
        logLine(String.format("  %s records updated", count));
    }

    public void onDeleteClick(View view) {

        logLine("[Delete]");

        if (!providerExists(Address.CONTENT_URI)) {
            logLine("  Content Provider が不在");
            return;
        }

        // ★ポイント 4 ★ センシティブな情報をリクエストに含めてはならない
        // リクエスト先のアプリがマルウェアである可能性がある。
        // マルウェアに取得されても問題のない情報であればリクエストに含めてもよい。
        int count = getContentResolver().delete(Address.CONTENT_URI, null, null);

        // ★ポイント 5 ★ 結果データの安全性を確認する
        // サンプルにつき割愛。「3.2 入力データの安全性を確認する」を参照。
        logLine(String.format("  %s records deleted", count));
```

(次のページに続く)

(前のページからの続き)

```
    }

    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 パートナー限定 Content Provider を作る・利用する

パートナー限定 Content Provider は、特定のアプリだけから利用できる Content Provider である。パートナー企業のアプリと自社アプリが連携してシステムを構成し、パートナーアプリとの間で扱う情報や機能を守るために利用される。

以下、パートナー限定 Content Provider の実装例を示す。

ポイント (Content Provider を作る) :

1. exported=”true” により、明示的に公開設定する
2. 利用元アプリの証明書がホワイトリストに登録されていることを確認する
3. パートナーアプリからのリクエストであっても、パラメータの安全性を確認する
4. パートナーアプリに開示してよい情報に限り返送してよい

```
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"
```

(次のページに続く)

(前のページからの続き)

```
      android:label="@string/app_name" >

    <!-- ★ポイント 1★ exported="true"により、明示的に公開設定する -->
    <provider
        android:name="org.jssec.android.provider.partnerprovider.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;
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;
```

(次のページに続く)

(前のページからの続き)

```
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";

    // Content Provider が提供するインターフェースを公開
    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);
    }

    // DB を使用せずに固定値を返す例にしているため、query メソッドで返す Cursor を事前に定義
    private static MatrixCursor sAddressCursor =
        new MatrixCursor(new String[] { "_id", "pref" });
    static {
        sAddressCursor.addRow(new String[] { "1", "北海道" });
        sAddressCursor.addRow(new String[] { "2", "青森" });
        sAddressCursor.addRow(new String[] { "3", "岩手" });
    }
```

(次のページに続く)

(前のページからの続き)

```
    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" });
    }

    // ★ポイント 2★ 利用元アプリの証明書がホワイトリストに登録されていることを確認する
    private static PkgCertWhitelists sWhitelists = null;
    private static void buildWhitelists(Context context) {
        boolean isdebug = Utils.isDebuggable(context);
        sWhitelists = new PkgCertWhitelists();

        // パートナーアプリ org.jssec.android.provider.partneruser の証明書ハッシュ値を登録
        sWhitelists.add("org.jssec.android.provider.partneruser", isdebug ?
            // debug.keystore の"androiddebugkey"の証明書ハッシュ値
            "0EFB7236 328348A9 89718BAD DF57F544 D5CCB4AE B9DB34BC 1E29DD26 F77C8255" :
            // keystore の"partner key"の証明書ハッシュ値
            "1F039BB5 7861C27A 3916C778 8E78CE00 690B3974 3EB8259F E2627B8D 4C0EC35A");

        // 以下同様に他のパートナーアプリを登録...
    }
    private static boolean checkPartner(Context context, String pkgname) {
        if (sWhitelists == null) buildWhitelists(context);
        return sWhitelists.test(context, pkgname);
    }
    // 利用元アプリのパッケージ名を取得
    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;
                    }
                }
            }
```

(次のページに続く)

(前のページからの続き)

```
        }

        return pkgname;
    }

    @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) {

        // ★ポイント 2★ 利用元アプリの証明書がホワイトリストに登録されていることを確認する
        if (!checkPartner(getContext(), getCallingPackage(getContext()))) {
            throw new SecurityException("利用元アプリはパートナーアプリではない。");
        }

        // ★ポイント 3★ パートナーアプリからのリクエストであっても、パラメータの安全性を確認する
        // ここでは uri が想定の範囲内であることを、UriMatcher#match() と switch case で確認している。
        // 「3.2 入力データの安全性を確認する」を参照。

        // ★ポイント 4★ パートナーアプリに開示してよい情報に限り返送してよい
        // query の結果がパートナーアプリに開示してよい情報かどうかはアプリ次第。
        switch (sUriMatcher.match(uri)) {
        case DOWNLOADS_CODE:
        case DOWNLOADS_ID_CODE:
```

(次のページに続く)

(前のページからの続き)

```
            return sDownloadCursor;

        case ADDRESSES_CODE:
        case ADDRESSES_ID_CODE:
            return sAddressCursor;

        default:
            throw new IllegalArgumentException("Invalid URI：" + uri);
        }
    }

    @Override
    public Uri insert(Uri uri, ContentValues values) {

        // ★ポイント 2★ 利用元アプリの証明書がホワイトリストに登録されていることを確認する
        if (!checkPartner(getContext(), getCallingPackage(getContext()))) {
            throw new SecurityException("利用元アプリはパートナーアプリではない。");
        }

        // ★ポイント 3★ パートナーアプリからのリクエストであっても、パラメータの安全性を確認する
        // ここでは uri が想定の範囲内であることを、UriMatcher#match() と switch case で確認している。
        // 「3.2 入力データの安全性を確認する」を参照。

        // ★ポイント 4★ パートナーアプリに開示してよい情報に限り返送してよい
        // Insert 結果、発番される ID がパートナーアプリに開示してよい情報かどうかはアプリ次第。
        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) {

        // ★ポイント 2★ 利用元アプリの証明書がホワイトリストに登録されていることを確認する
        if (!checkPartner(getContext(), getCallingPackage(getContext()))) {
            throw new SecurityException("利用元アプリはパートナーアプリではない。");
        }
```

(次のページに続く)

(前のページからの続き)

```
        // ★ポイント 3★ パートナーアプリからのリクエストであっても、パラメータの安全性を確認する
        // ここでは uri が想定の範囲内であることを、UriMatcher#match() と switch case で確認している。
        // 「3.2 入力データの安全性を確認する」を参照。

        // ★ポイント 4★ パートナーアプリに開示してよい情報に限り返送してよい
        // Update されたレコード数がセンシティブな意味を持つかどうかはアプリ次第。
        switch (sUriMatcher.match(uri)) {
        case DOWNLOADS_CODE:
            return 5;       // update されたレコード数を返す

        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) {

        // ★ポイント 2★ 利用元アプリの証明書がホワイトリストに登録されていることを確認する
        if (!checkPartner(getContext(), getCallingPackage(getContext()))) {
            throw new SecurityException("利用元アプリはパートナーアプリではない。");
        }

        // ★ポイント 3★ パートナーアプリからのリクエストであっても、パラメータの安全性を確認する
        // ここでは uri が想定の範囲内であることを、UriMatcher#match() と switch case で確認している。
        // 「3.2 入力データの安全性を確認する」を参照。

        // ★ポイント 4★ パートナーアプリに開示してよい情報に限り返送してよい
        // Delete されたレコード数がセンシティブな意味を持つかどうかはアプリ次第。
        switch (sUriMatcher.match(uri)) {
        case DOWNLOADS_CODE:
            return 10;      // delete されたレコード数を返す

        case DOWNLOADS_ID_CODE:
            return 1;
```

(次のページに続く)

(前のページからの続き)

```
        case ADDRESSES_CODE:
            return 20;

        case ADDRESSES_ID_CODE:
            return 1;

        default:
            throw new IllegalArgumentException("Invalid URI:" + uri);
        }
    }
}
```

次に、パートナー限定 Content Provider を利用する Activity の例を示す。

ポイント (Content Provider を利用する):

5. 利用先パートナー限定 Content Provider アプリの証明書がホワイトリストに登録されていることを確認する
6. パートナー限定 Content Provider アプリに開示してよい情報に限りリクエストに含めてよい
7. パートナー限定 Content Provider アプリからの結果であっても、結果データの安全性を確認する

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;
```

(次のページに続く)

(前のページからの続き)

```
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 {

    // 利用先の Content Provider 情報
    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);
    }

    // ★ポイント 5★ 利用先パートナー限定 Content Provider アプリの証明書がホワイトリストに登録されている
    // ことを確認する
    private static PkgCertWhitelists sWhitelists = null;
    private static void buildWhitelists(Context context) {
        boolean isdebug = Utils.isDebuggable(context);
        sWhitelists = new PkgCertWhitelists();

        // パートナー限定 Content Provider アプリ org.jssec.android.provider.partnerprovider の証明書
        // ハッシュ値を登録
        sWhitelists.add("org.jssec.android.provider.partnerprovider", isdebug ?
            // debug.keystore の"androiddebugkey"の証明書ハッシュ値
            "0EFB7236 328348A9 89718BAD DF57F544 D5CCB4AE B9DB34BC 1E29DD26 F77C8255" :
            // keystore の"my company key"の証明書ハッシュ値
            "D397D343 A5CBC10F 4EDDEB7C A10062DE 5690984F 1FB9E88B D7B3A7C2 42E142CA");

        // 以下同様に他のパートナー限定 Content Provider アプリを登録...
    }
    private static boolean checkPartner(Context context, String pkgname) {
        if (sWhitelists == null) buildWhitelists(context);
        return sWhitelists.test(context, pkgname);
    }
    // uri を AUTHORITY とする Content Provider のパッケージ名を取得
    private String providerPkgname(Uri uri) {
        String pkgname = null;
        ProviderInfo pi =
```

(次のページに続く)

(前のページからの続き)

```
            getPackageManager().resolveContentProvider(uri.getAuthority(), 0);
        if (pi != null) pkgname = pi.packageName;
        return pkgname;
    }

    public void onQueryClick(View view) {

        logLine("[Query]");

        // ★ポイント 5★ 利用先パートナー限定 Content Provider アプリの証明書がホワイトリストに登録されている
        // ことを確認する
        if (!checkPartner(this, providerPkgname(Address.CONTENT_URI))) {
            logLine("  利用先 Content Provider アプリはホワイトリストに登録されていない。");
            return;
        }

        // ★ポイント 6★ パートナー限定 Content Provider アプリに開示してよい情報に限りリクエストに含めてよい
        Cursor cursor = null;
        try {
            cursor =
                getContentResolver().query(Address.CONTENT_URI, null, null, null, null);

            // ★ポイント 7★ パートナー限定 Content Provider アプリからの結果であっても、結果データの安全性を
            // 確認する
            // サンプルにつき割愛。「3.2 入力データの安全性を確認する」を参照。
            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();
        }
    }
```

(次のページに続く)

(前のページからの続き)

```
    public void onInsertClick(View view) {

        logLine("[Insert]");

        // ★ポイント 5★ 利用先パートナー限定 Content Provider アプリの証明書がホワイトリストに登録されている
        // ことを確認する
        if (!checkPartner(this, providerPkgname(Address.CONTENT_URI))) {
            logLine("  利用先 Content Provider アプリはホワイトリストに登録されていない。");
            return;
        }

        // ★ポイント 6★ パートナー限定 Content Provider アプリに開示してよい情報に限りリクエストに含めてよい
        ContentValues values = new ContentValues();
        values.put("pref", "東京都");
        Uri uri = getContentResolver().insert(Address.CONTENT_URI, values);

        // ★ポイント 7★ パートナー限定 Content Provider アプリからの結果であっても、結果データの安全性を
        // 確認する
        // サンプルにつき割愛。「3.2 入力データの安全性を確認する」を参照。
        logLine("  uri:" + uri);
    }

    public void onUpdateClick(View view) {

        logLine("[Update]");

        // ★ポイント 5★ 利用先パートナー限定 Content Provider アプリの証明書がホワイトリストに登録されている
        // ことを確認する
        if (!checkPartner(this, providerPkgname(Address.CONTENT_URI))) {
            logLine("  利用先 Content Provider アプリはホワイトリストに登録されていない。");
            return;
        }

        // ★ポイント 6★ パートナー限定 Content Provider アプリに開示してよい情報に限りリクエストに含めてよい
        ContentValues values = new ContentValues();
        values.put("pref", "東京都");
        String where = "_id = ?";
        String[] args = { "4" };
        int count = getContentResolver().update(Address.CONTENT_URI, values, where, args);
```

(次のページに続く)

(前のページからの続き)

```
        // ★ポイント 7★ パートナー限定 Content Provider アプリからの結果であっても、結果データの安全性を
        // 確認する
        // サンプルにつき割愛。「3.2 入力データの安全性を確認する」を参照。
        logLine(String.format("  %s records updated", count));
    }

    public void onDeleteClick(View view) {

        logLine("[Delete]");

        // ★ポイント 5★ 利用先パートナー限定 Content Provider アプリの証明書がホワイトリストに登録されている
        // ことを確認する
        if (!checkPartner(this, providerPkgname(Address.CONTENT_URI))) {
            logLine("  利用先 Content Provider アプリはホワイトリストに登録されていない。");
            return;
        }

        // ★ポイント 6★ パートナー限定 Content Provider アプリに開示してよい情報に限りリクエストに含めてよい
        int count = getContentResolver().delete(Address.CONTENT_URI, null, null);

        // ★ポイント 7★ パートナー限定 Content Provider アプリからの結果であっても、結果データの安全性を
        // 確認する
        // サンプルにつき割愛。「3.2 入力データの安全性を確認する」を参照。
        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

```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 は 32 バイト
        sha256 = sha256.toUpperCase();
        if (sha256.replaceAll("[0-9A-F]+", "").length() != 0)
            return false;    // 0-9A-F 以外の文字がある

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

(次のページに続く)

(前のページからの続き)

```
    public boolean test(Context ctx, String pkgname) {
        // pkgname に対応する正解のハッシュ値を取得する
        String correctHash = mWhitelists.get(pkgname);

        // pkgname の実際のハッシュ値と正解のハッシュ値を比較する
        if (Build.VERSION.SDK_INT >= 28) {
            // ★ API Level >= 28 では Package Manager の API で直接検証が可能
            PackageManager pm = ctx.getPackageManager();
            return pm.hasSigningCertificate(pkgname,
                                            Utils.hex2Bytes(correctHash),
                                            CERT_INPUT_SHA256);
        } else {
            // API Level < 28 の場合は PkgCert の機能を利用する
            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;
```

(次のページに続く)

(前のページからの続き)

```
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);
            if (pkginfo.signatures.length != 1) return null;  // 複数署名は扱わない
            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();
    }
}
```

4.3.1.4 自社限定 **Content Provider** を作る・利用する

自社限定 Content Provider は、自社以外のアプリから利用されることを禁止する Content Provider である。複数の自社製アプリでシステムを構成し、自社アプリが扱う情報や機能を守るために利用される。

以下、自社限定 Content Provider の実装例を示す。

ポイント (Content Provider を作る) :

1. 独自定義 Signature Permission を定義する
2. 独自定義 Signature Permission を要求宣言する
3. exported=”true” により、明示的に公開設定する
4. 独自定義 Signature Permission が自社アプリにより定義されていることを確認する
5. 自社アプリからのリクエストであっても、パラメータの安全性を確認する
6. 利用元アプリは自社アプリであるから、センシティブな情報を返送してよい
7. APK を Export するときに、利用元アプリと同じ開発者鍵で APK を署名する

AndroidManifest.xml

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

  <!-- ★ポイント 1★ 独自定義 Signature Permission を定義する -->
  <permission
      android:name="org.jssec.android.provider.inhouseprovider.MY_PERMISSION"
      android:protectionLevel="signature" />

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

    <!-- ★ポイント 2★ 独自定義 Signature Permission を要求宣言する -->
    <!-- ★ポイント 3★ exported="true"により、明示的に公開設定する -->
    <provider
        android:name="org.jssec.android.provider.inhouseprovider.InhouseProvider"
        android:authorities="org.jssec.android.provider.inhouseprovider"
        android:permission="org.jssec.android.provider.inhouseprovider.MY_PERMISSION"
        android:exported="true"/>
  </application>
</manifest>
```

InhouseProvider.java

```
/*
 * Copyright (C) 2012-2026 Japan Smartphone Security Association
 *
```

(次のページに続く)

(前のページからの続き)

```
 * 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";

    // Content Provider が提供するインターフェースを公開
    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 =
```

(次のページに続く)

(前のページからの続き)

```
            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);
    }

    // DB を使用せずに固定値を返す例にしているため、query メソッドで返す Cursor を事前に定義
    private static MatrixCursor sAddressCursor =
        new MatrixCursor(new String[] { "_id", "pref" });
    static {
        sAddressCursor.addRow(new String[] { "1", "北海道" });
        sAddressCursor.addRow(new String[] { "2", "青森" });
        sAddressCursor.addRow(new String[] { "3", "岩手" });
    }
    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" });
    }

    // 自社の Signature Permission
    private static final String MY_PERMISSION =
        "org.jssec.android.provider.inhouseprovider.MY_PERMISSION";

    // 自社の証明書のハッシュ値
    private static String sMyCertHash = null;
    private static String myCertHash(Context context) {
        if (sMyCertHash == null) {
            if (Utils.isDebuggable(context)) {
                // debug.keystore の"androiddebugkey"の証明書ハッシュ値
                sMyCertHash = "0EFB7236 328348A9 89718BAD DF57F544 D5CCB4AE B9DB34BC␣
↪1E29DD26 F77C8255";
            } else {
```

(次のページに続く)

(前のページからの続き)

```
                // keystore の"my company key"の証明書ハッシュ値
                sMyCertHash = "D397D343 A5CBC10F 4EDDEB7C A10062DE 5690984F 1FB9E88B␣
↪D7B3A7C2 42E142CA";
            }
        }
        return sMyCertHash;
    }

    @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) {

        // ★ポイント 4★ 独自定義 Signature Permission が自社アプリにより定義されていることを確認する
        if (!SigPerm.test(getContext(), MY_PERMISSION, myCertHash(getContext()))) {
            throw new SecurityException("独自定義 Signature Permission が自社アプリにより定義されていない。");
        }

        // ★ポイント 5★ 自社アプリからのリクエストであっても、パラメータの安全性を確認する
        // ここでは uri が想定の範囲内であることを、UriMatcher#match() と switch case で確認している。
        // 「3.2 入力データの安全性を確認する」を参照。
        // ★ポイント 6★ 利用元アプリは自社アプリであるから、センシティブな情報を返送してよい
        // query の結果が自社アプリに開示してよい情報かどうかはアプリ次第。
```

(次のページに続く)

(前のページからの続き)

```
        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
    public Uri insert(Uri uri, ContentValues values) {

        // ★ポイント 4★ 独自定義 Signature Permission が自社アプリにより定義されていることを確認する
        if (!SigPerm.test(getContext(), MY_PERMISSION, myCertHash(getContext()))) {
            throw new SecurityException("独自定義 Signature Permission が自社アプリにより定義されていない。");
        }

        // ★ポイント 5★ 自社アプリからのリクエストであっても、パラメータの安全性を確認する
        // ここでは uri が想定の範囲内であることを、UriMatcher#match() と switch case で確認している。
        // 「3.2 入力データの安全性を確認する」を参照。
        // ★ポイント 6★ 利用元アプリは自社アプリであるから、センシティブな情報を返送してよい
        // Insert 結果、発番される ID が自社アプリに開示してよい情報かどうかはアプリ次第。
        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) {

        // ★ポイント 4★ 独自定義 Signature Permission が自社アプリにより定義されていることを確認する
```

(次のページに続く)

(前のページからの続き)

```
        if (!SigPerm.test(getContext(), MY_PERMISSION, myCertHash(getContext()))) {
            throw new SecurityException("独自定義 Signature Permission が自社アプリにより定義されていない。");
        }

        // ★ポイント 5★ 自社アプリからのリクエストであっても、パラメータの安全性を確認する
        // ここでは uri が想定の範囲内であることを、UriMatcher#match() と switch case で確認している。
        // 「3.2 入力データの安全性を確認する」を参照。
        // ★ポイント 6★ 利用元アプリは自社アプリであるから、センシティブな情報を返送してよい
        // Update されたレコード数がセンシティブな意味を持つかどうかはアプリ次第。
        switch (sUriMatcher.match(uri)) {
        case DOWNLOADS_CODE:
            return 5;    // update されたレコード数を返す

        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) {

        // ★ポイント 4★ 独自定義 Signature Permission が自社アプリにより定義されていることを確認する
        if (!SigPerm.test(getContext(), MY_PERMISSION, myCertHash(getContext()))) {
            throw new SecurityException("独自定義 Signature Permission が自社アプリにより定義されていない。");
        }

        // ★ポイント 5★ 自社アプリからのリクエストであっても、パラメータの安全性を確認する
        // ここでは uri が想定の範囲内であることを、UriMatcher#match() と switch case で確認している。
        // 「3.2 入力データの安全性を確認する」を参照。
        // ★ポイント 6★ 利用元アプリは自社アプリであるから、センシティブな情報を返送してよい
        // Delete されたレコード数がセンシティブな意味を持つかどうかはアプリ次第。
        switch (sUriMatcher.match(uri)) {
        case DOWNLOADS_CODE:
            return 10;  // delete されたレコード数を返す
```

(次のページに続く)

(前のページからの続き)

```
        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

```
/*
 * 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 {
```

(次のページに続く)

(前のページからの続き)

```
    public static boolean test(Context ctx, String sigPermName, String correctHash) {
        if (correctHash == null) return false;
        correctHash = correctHash.replaceAll(" ", "");
        try{
            // sigPermName を定義したアプリのパッケージ名を取得する
            PackageManager pm = ctx.getPackageManager();
            PermissionInfo pi =
                pm.getPermissionInfo(sigPermName, PackageManager.GET_META_DATA);
            String pkgname = pi.packageName;
            // 非 Signature Permission の場合は失敗扱い
            if (pi.protectionLevel != PermissionInfo.PROTECTION_SIGNATURE) return false;
            // pkgname の実際のハッシュ値と正解のハッシュ値を比較する
            if (Build.VERSION.SDK_INT >= 28) {
                // ★ API Level >= 28 では Package Manager の API で直接検証が可能
                return pm.hasSigningCertificate(pkgname,
                                                Utils.hex2Bytes(correctHash),
                                                CERT_INPUT_SHA256);
            } else {
                // API Level < 28 の場合は PkgCert を利用し、ハッシュ値を取得して比較する
                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.
 */
```

(次のページに続く)

(前のページからの続き)

```
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);
            if (pkginfo.signatures.length != 1) return null;  // 複数署名は扱わない
            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) {
```

(次のページに続く)

(前のページからの続き)

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

★ポイント 7 ★　APK を Export するときに、利用元アプリと同じ開発者鍵で APK を署名する。

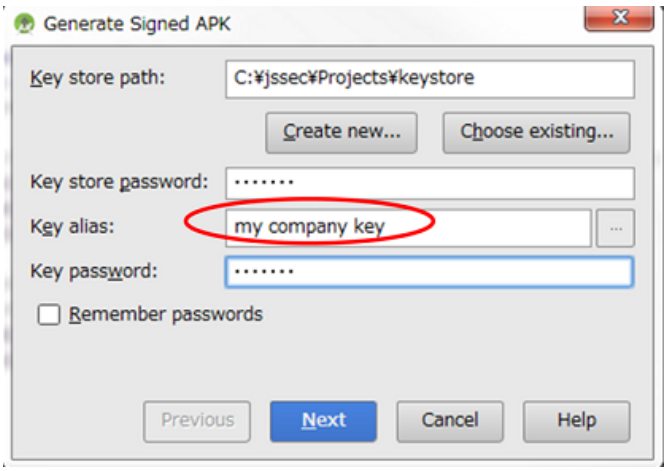

図 4.3.2　利用元アプリと同じ開発者鍵で APK を署名する

次に、自社限定 Content Provider を利用する Activity の例を示す。

ポイント (Content Provider を利用する) :

8. 独自定義 Signature Permission を利用宣言する
9. 独自定義 Signature Permission が自社アプリにより定義されていることを確認する
10. 利用先 Content Provider アプリの証明書が自社の証明書であることを確認する
11. 自社限定 Content Provider アプリに開示してよい情報に限りリクエストに含めてよい
12. 自社限定 Content Provider アプリからの結果であっても、結果データの安全性を確認する
13. APK を Export するときに、利用先アプリと同じ開発者鍵で APK を署名する

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.provider.inhouseprovider" />
  </queries>
```

(次のページに続く)

(前のページからの続き)

```
  <!-- ★ポイント 8 ★ 独自定義 Signature Permission を利用宣言する -->
  <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="org.jssec.android.provider.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>
  </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;
```

(次のページに続く)

(前のページからの続き)

```
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 {

    // 利用先の Content Provider 情報
    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);
    }

    // 自社の Signature Permission
    private static final String MY_PERMISSION =
        "org.jssec.android.provider.inhouseprovider.MY_PERMISSION";

    // 自社の証明書のハッシュ値
    private static String sMyCertHash = null;
    private static String myCertHash(Context context) {
        if (sMyCertHash == null) {
            if (Utils.isDebuggable(context)) {
                // debug.keystore の"androiddebugkey"の証明書ハッシュ値
                sMyCertHash = "0EFB7236 328348A9 89718BAD DF57F544 D5CCB4AE B9DB34BC␣
↪1E29DD26 F77C8255";
            } else {
                // keystore の"my company key"の証明書ハッシュ値
                sMyCertHash = "D397D343 A5CBC10F 4EDDEB7C A10062DE 5690984F 1FB9E88B␣
↪D7B3A7C2 42E142CA";
            }
        }
        return sMyCertHash;
    }

    // 利用先 Content Provider のパッケージ名を取得
    private static String providerPkgname(Context context, Uri uri) {
        String pkgname = null;
        PackageManager pm = context.getPackageManager();
```

(次のページに続く)

(前のページからの続き)

```
        ProviderInfo pi = pm.resolveContentProvider(uri.getAuthority(), 0);
        if (pi != null) pkgname = pi.packageName;
        return pkgname;
    }

    public void onQueryClick(View view) {

        logLine("[Query]");

        // ★ポイント 9★ 独自定義 Signature Permission が自社アプリにより定義されていることを確認する
        if (!SigPerm.test(this, MY_PERMISSION, myCertHash(this))) {
            logLine("  独自定義 Signature Permission が自社アプリにより定義されていない。");
            return;
        }

        // ★ポイント 10★ 利用先 Content Provider アプリの証明書が自社の証明書であることを確認する
        String pkgname = providerPkgname(this, Address.CONTENT_URI);
        if (!PkgCert.test(this, pkgname, myCertHash(this))) {
            logLine("  利用先 Content Provider は自社アプリではない。");
            return;
        }

        // ★ポイント 11★ 自社限定 Content Provider アプリに開示してよい情報に限りリクエストに含めてよい
        Cursor cursor = null;
        try {
            cursor =
                getContentResolver().query(Address.CONTENT_URI, null, null, null, null);

            // ★ポイント 12★ 自社限定 Content Provider アプリからの結果であっても、結果データの安全性を確認する
            // サンプルにつき割愛。「3.2 入力データの安全性を確認する」を参照。
            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 {
```

(次のページに続く)

(前のページからの続き)

```
            if (cursor != null) cursor.close();
        }
    }

    public void onInsertClick(View view) {

        logLine("[Insert]");

        // ★ポイント 9★ 独自定義 Signature Permission が自社アプリにより定義されていることを確認する
        String correctHash = myCertHash(this);
        if (!SigPerm.test(this, MY_PERMISSION, correctHash)) {
            logLine("  独自定義 Signature Permission が自社アプリにより定義されていない。");
            return;
        }

        // ★ポイント 10★ 利用先 Content Provider アプリの証明書が自社の証明書であることを確認する
        String pkgname = providerPkgname(this, Address.CONTENT_URI);
        if (!PkgCert.test(this, pkgname, correctHash)) {
            logLine("  利用先 Content Provider は自社アプリではない。");
            return;
        }

        // ★ポイント 11★ 自社限定 Content Provider アプリに開示してよい情報に限りリクエストに含めてよい
        ContentValues values = new ContentValues();
        values.put("pref", "東京都");
        Uri uri = getContentResolver().insert(Address.CONTENT_URI, values);

        // ★ポイント 12★ 自社限定 Content Provider アプリからの結果であっても、結果データの安全性を確認する
        // サンプルにつき割愛。「3.2 入力データの安全性を確認する」を参照。
        logLine("  uri:" + uri);
    }

    public void onUpdateClick(View view) {

        logLine("[Update]");

        // ★ポイント 9★ 独自定義 Signature Permission が自社アプリにより定義されていることを確認する
        String correctHash = myCertHash(this);
        if (!SigPerm.test(this, MY_PERMISSION, correctHash)) {
            logLine("  独自定義 Signature Permission が自社アプリにより定義されていない。");
            return;
        }
```

(次のページに続く)

(前のページからの続き)

```
        // ★ポイント 10★ 利用先 Content Provider アプリの証明書が自社の証明書であることを確認する
        String pkgname = providerPkgname(this, Address.CONTENT_URI);
        if (!PkgCert.test(this, pkgname, correctHash)) {
            logLine("  利用先 Content Provider は自社アプリではない。");
            return;
        }

        // ★ポイント 11★ 自社限定 Content Provider アプリに開示してよい情報に限りリクエストに含めてよい
        ContentValues values = new ContentValues();
        values.put("pref", "東京都");
        String where = "_id = ?";
        String[] args = { "4" };
        int count = getContentResolver().update(Address.CONTENT_URI, values, where, args);

        // ★ポイント 12★ 自社限定 Content Provider アプリからの結果であっても、結果データの安全性を確認する
        // サンプルにつき割愛。「3.2 入力データの安全性を確認する」を参照。
        logLine(String.format("  %s records updated", count));
    }

    public void onDeleteClick(View view) {

        logLine("[Delete]");

        // ★ポイント 9★ 独自定義 Signature Permission が自社アプリにより定義されていることを確認する
        String correctHash = myCertHash(this);
        if (!SigPerm.test(this, MY_PERMISSION, correctHash)) {
            logLine("  独自定義 Signature Permission が自社アプリにより定義されていない。");
            return;
        }

        // ★ポイント 10★ 利用先 Content Provider アプリの証明書が自社の証明書であることを確認する
        String pkgname = providerPkgname(this, Address.CONTENT_URI);
        if (!PkgCert.test(this, pkgname, correctHash)) {
            logLine("  利用先 Content Provider は自社アプリではない。");
            return;
        }

        // ★ポイント 11★ 自社限定 Content Provider アプリに開示してよい情報に限りリクエストに含めてよい
        int count = getContentResolver().delete(Address.CONTENT_URI, null, null);

        // ★ポイント 12★ 自社限定 Content Provider アプリからの結果であっても、結果データの安全性を確認する
```

(次のページに続く)

(前のページからの続き)

```
        // サンプルにつき割愛。「3.2 入力データの安全性を確認する」を参照。
        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
 * 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;
```

(次のページに続く)

(前のページからの続き)

```
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{
            // sigPermName を定義したアプリのパッケージ名を取得する
            PackageManager pm = ctx.getPackageManager();
            PermissionInfo pi =
                pm.getPermissionInfo(sigPermName, PackageManager.GET_META_DATA);
            String pkgname = pi.packageName;
            // 非 Signature Permission の場合は失敗扱い
            if (pi.protectionLevel != PermissionInfo.PROTECTION_SIGNATURE) return false;
            // pkgname の実際のハッシュ値と正解のハッシュ値を比較する
            if (Build.VERSION.SDK_INT >= 28) {
                // ★ API Level >= 28 では Package Manager の API で直接検証が可能
                return pm.hasSigningCertificate(pkgname,
                                                Utils.hex2Bytes(correctHash),
                                                CERT_INPUT_SHA256);
            } else {
                // API Level < 28 の場合は PkgCert を利用し、ハッシュ値を取得して比較する
                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.
```

(次のページに続く)

(前のページからの続き)

```
 * 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);
            if (pkginfo.signatures.length != 1) return null;  // 複数署名は扱わない
            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;
        }
```

(次のページに続く)

(前のページからの続き)

```
    }

    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();
    }
}
```

★ポイント 13 ★ APK を Export するときに、利用先アプリと同じ開発者鍵で APK を署名する。

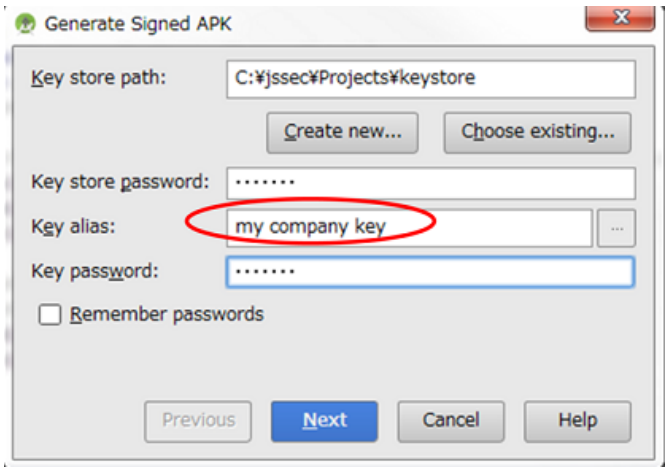

図 4.3.3　利用先アプリと同じ開発者鍵で APK を署名する

4.3.1.5 一時許可 Content Provider を作る・利用する

一時許可 Content Provider は、基本的には非公開の Content Provider であるが、特定のアプリに対して一時的に特定 URI へのアクセスを許可する Content Provider である。特殊なフラグを指定した Intent を対象アプリに送付することにより、そのアプリに一時的なアクセス権限が付されるようになっている。Content Provider 側アプリが能動的に他のアプリにアクセス許可を与えることもできるし、一時的なアクセス許可を求めてきたアプリに Content Provider 側アプリが受動的にアクセス許可を与えることもできる。

以下、一時許可 Content Provider の実装例を示す。

ポイント (Content Provider を作る) :

1. exported=”false” により、一時許可する Path 以外を非公開設定する
2. grant-uri-permission により、一時許可する Path を指定する
3. 一時的に許可したアプリからのリクエストであっても、パラメータの安全性を確認する
4. 一時的に許可したアプリに開示してよい情報に限り返送してよい
5. 一時的にアクセスを許可する URI を Intent に指定する

6. 一時的に許可するアクセス権限を Intent に指定する
7. 一時的にアクセスを許可するアプリに明示的 Intent を送信する
8. 一時許可の要求元アプリに Intent を返信する

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=".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>

    <!-- 一時許可 Content Provider -->
    <!-- ★ポイント 1★ exported="false"により、一時許可する Path 以外を非公開設定する -->
    <provider
        android:name=".TemporaryProvider"
        android:authorities="org.jssec.android.provider.temporaryprovider"
        android:exported="false" >

      <!-- ★ポイント 2★ grant-uri-permission により、一時許可する Path を指定する -->
      <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
```

(次のページに続く)

(前のページからの続き)

```
 *
 * 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;
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";

    // Content Providerが提供するインターフェースを公開
    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
```

(次のページに続く)

(前のページからの続き)

```
    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);
    }

    // DB を使用せずに固定値を返す例にしているため、query メソッドで返す Cursor を事前に定義
    private static MatrixCursor sAddressCursor =
        new MatrixCursor(new String[] { "_id", "pref" });
    static {
        sAddressCursor.addRow(new String[] { "1", "北海道" });
        sAddressCursor.addRow(new String[] { "2", "青森" });
        sAddressCursor.addRow(new String[] { "3", "岩手" });
    }
    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;
```

(次のページに続く)

(前のページからの続き)

```
        default:
            throw new IllegalArgumentException("Invalid URI:" + uri);
        }
    }

    @Override
    public Cursor query(Uri uri, String[] projection, String selection,
                        String[] selectionArgs, String sortOrder) {

        // ★ポイント 3★ 一時的に許可したアプリからのリクエストであっても、パラメータの安全性を確認する
        // ここでは uri が想定の範囲内であることを、UriMatcher#match() と switch case で確認している。
        // その他のパラメータの確認はサンプルにつき省略。「3.2 入力データの安全性を確認する」を参照。
        // ★ポイント 4★ 一時的に許可したアプリに開示してよい情報に限り返送してよい
        // query の結果がセンシティブな意味を持つかどうかはアプリ次第。
        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
    public Uri insert(Uri uri, ContentValues values) {

        // ★ポイント 3★ 一時的に許可したアプリからのリクエストであっても、パラメータの安全性を確認する
        // ここでは uri が想定の範囲内であることを、UriMatcher#match() と switch case で確認している。
        // その他のパラメータの確認はサンプルにつき省略。「3.2 入力データの安全性を確認する」を参照。
        // ★ポイント 4★ 一時的に許可したアプリに開示してよい情報に限り返送してよい
        // Insert 結果、発番される ID がセンシティブな意味を持つかどうかはアプリ次第。
        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);
```

(次のページに続く)

(前のページからの続き)

```
    }
}

@Override
public int update(Uri uri, ContentValues values, String selection,
                  String[] selectionArgs) {

    // ★ポイント 3★ 一時的に許可したアプリからのリクエストであっても、パラメータの安全性を確認する
    // ここでは uri が想定の範囲内であることを、UriMatcher#match() と switch case で確認している。
    // その他のパラメータの確認はサンプルにつき省略。「3.2 入力データの安全性を確認する」を参照。
    // ★ポイント 4★ 一時的に許可したアプリに開示してよい情報に限り返送してよい
    // Update されたレコード数がセンシティブな意味を持つかどうかはアプリ次第。
    switch (sUriMatcher.match(uri)) {
    case DOWNLOADS_CODE:
        return 5;    // update されたレコード数を返す

    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) {

    // ★ポイント 3★ 一時的に許可したアプリからのリクエストであっても、パラメータの安全性を確認する
    // ここでは uri が想定の範囲内であることを、UriMatcher#match() と switch case で確認している。
    // その他のパラメータの確認はサンプルにつき省略。「3.2 入力データの安全性を確認する」を参照。
    // ★ポイント 4★ 一時的に許可したアプリに開示してよい情報に限り返送してよい
    // Delete されたレコード数がセンシティブな意味を持つかどうかはアプリ次第。
    switch (sUriMatcher.match(uri)) {
    case DOWNLOADS_CODE:
        return 10;  // delete されたレコード数を返す

    case DOWNLOADS_ID_CODE:
        return 1;
```

(次のページに続く)

(前のページからの続き)

```
        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;

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に関する情報
    private static final String TARGET_PACKAGE =
        "org.jssec.android.provider.temporaryuser";
    private static final String TARGET_ACTIVITY =
```

(次のページに続く)

(前のページからの続き)

```
        "org.jssec.android.provider.temporaryuser.TemporaryUserActivity";

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

    // Content Provider 側アプリが能動的に他のアプリにアクセス許可を与えるケース
    public void onSendClick(View view) {
        try {
            Intent intent = new Intent();

            // ★ポイント 5★ 一時的にアクセスを許可する URI を Intent に指定する
            intent.setData(TemporaryProvider.Address.CONTENT_URI);

            // ★ポイント 6★ 一時的に許可するアクセス権限を Intent に指定する
            intent.setFlags(Intent.FLAG_GRANT_READ_URI_PERMISSION);

            // ★ポイント 7★ 一時的にアクセスを許可するアプリに明示的 Intent を送信する
            intent.setClassName(TARGET_PACKAGE, TARGET_ACTIVITY);
            startActivity(intent);

        } catch (ActivityNotFoundException e) {
            Toast.makeText(this, "User Activity が見つからない。", 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
 *
 *      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.
```

(次のページに続く)

(前のページからの続き)

```
 */

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);
    }

    // 一時的なアクセス許可を求めてきたアプリに Content Provider 側アプリが受動的にアクセス許可を与えるケース
    public void onGrantClick(View view) {
        Intent intent = new Intent();

        // ★ポイント 5★ 一時的にアクセスを許可する URI を Intent に指定する
        intent.setData(TemporaryProvider.Address.CONTENT_URI);

        // ★ポイント 6★ 一時的に許可するアクセス権限を Intent に指定する
        intent.setFlags(Intent.FLAG_GRANT_READ_URI_PERMISSION);

        // ★ポイント 8★ 一時許可の要求元アプリに Intent を返信する
        setResult(Activity.RESULT_OK, intent);
        finish();
    }

    public void onCloseClick(View view) {
        finish();
    }
}
```

次に、一時許可 Content Provider を利用する Activity の例を示す。

ポイント (Content Provider を利用する) :

9. センシティブな情報をリクエストに含めてはならない
10. 結果データの安全性を確認する

TemporaryUserActivity.java

(次のページに続く)

(前のページからの続き)

```
/*
 * 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.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;
import android.widget.Toast;

public class TemporaryUserActivity extends Activity {

    // Provider Activity に関する情報
    private static final String TARGET_PACKAGE =
        "org.jssec.android.provider.temporaryprovider";
    private static final String TARGET_ACTIVITY =
        "org.jssec.android.provider.temporaryprovider.TemporaryPassiveGrantActivity";

    // 利用先の Content Provider 情報
    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);
    }
```

(次のページに続く)

(前のページからの続き)

```
    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 が不在");
                return;
            }

            // ★ポイント 9★ センシティブな情報をリクエストに含めてはならない
            // リクエスト先のアプリがマルウェアである可能性がある。
            // マルウェアに取得されても問題のない情報であればリクエストに含めてもよい。
            cursor = getContentResolver()
                    .query(Address.CONTENT_URI, null, null, null, null);

            // ★ポイント 10★ 結果データの安全性を確認する
            // サンプルにつき割愛。「3.2 入力データの安全性を確認する」を参照。
            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("  例外:" + ex.getMessage());
        }
        finally {
            if (cursor != null) cursor.close();
        }
    }

    // このアプリが一時的なアクセス許可を要求し、Content Provider 側アプリが受動的にアクセス許可を与えるケース
    public void onGrantRequestClick(View view) {
        Intent intent = new Intent();
        intent.setClassName(TARGET_PACKAGE, TARGET_ACTIVITY);
```

(次のページに続く)

(前のページからの続き)

```
        try {
            startActivityForResult(intent, REQUEST_CODE);
        } catch (ActivityNotFoundException e) {
            logLine("Grant の要求に失敗しました。\nTemporaryProvider がインストールされているか確認してください。");
        }
    }

    private boolean providerExists(Uri uri) {
        ProviderInfo pi =
            getPackageManager().resolveContentProvider(uri.getAuthority(), 0);
        return (pi != null);
    }

    private TextView mLogView;

    // Content Provider 側アプリが能動的にこのアプリにアクセス許可を与えるケース
    @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");
    }
}
```

4.3.2 ルールブック

Content Provider の実装時には以下のルールを守ること。

1. アプリ内でのみ使用する *Content Provider* は非公開設定する（必須）
2. リクエストパラメータの安全性を確認する（必須）
3. 独自定義 *Signature Permission* は、自社アプリが定義したことを確認して利用する（必須）
4. 結果情報を返す場合には、返送先アプリからの結果情報漏洩に注意する（必須）
5. 資産を二次的に提供する場合には、その資産の従来の保護水準を維持する（必須）

また、利用側は、以下のルールも守ること。

6. *Content Provider* の結果データの安全性を確認する（必須）

4.3.2.1 アプリ内でのみ使用する **Content Provider** は非公開設定する（必須）

同一アプリ内からのみ利用される Content Provider は他のアプリからアクセスできる必要がないだけでなく、開発者も Content Provider を攻撃するアクセスを考慮しないことが多い。Content Provider はデータ共有するための仕組みであるため、デフォルトでは公開扱いになってしまう。同一アプリ内からのみ利用される Content Provider は明示的に非公開設定し、非公開 Content Provider とすべきである。

AndroidManifest.xml

```
    <!-- ★ 4.3.1.1 - ポイント 1 ★ exported="false"により、明示的に非公開設定する -->
    <provider
        android:name=".PrivateProvider"
        android:authorities="org.jssec.android.provider.privateprovider"
        android:exported="false" />
```

4.3.2.2 リクエストパラメータの安全性を確認する（必須）

Content Provider のタイプによって若干リスクは異なるが、リクエストパラメータを処理する際には、まずその安全性を確認しなければならない。

Content Provider の各メソッドは SQL 文の構成要素パラメータを受け取ることを想定したインターフェースになってはいるものの、仕組みの上では単に任意の文字列を受け渡すだけのものであり、Content Provider 側では想定外のパラメータが与えられるケースを想定しなければならないことに注意が必要だ。

公開 Content Provider は不特定多数のアプリからリクエストを受け取るため、マルウェアの攻撃リクエストを受け取る可能性がある。非公開 Content Provider は他のアプリからリクエストを直接受け取ることはない。しかし同一アプリ内の公開 Activity が他のアプリから受け取った Intent のデータを非公開 Content Provider に転送するといったケースも考えられるため、リクエストを無条件に安全であると考えてはならない。その他の Content Provider についても、やはりリクエストの安全性を確認する必要がある。

「3.2. 入力データの安全性を確認する」を参照すること。

4.3.2.3 独自定義 **Signature Permission** は、自社アプリが定義したことを確認して利用する（必須）

自社アプリだけから利用できる自社限定 Content Provider を作る場合、独自定義 Signature Permission により保護しなければならない。AndroidManifest.xml での Permission 定義、Permission 要求宣言だけでは保護が不十分であるため、「5.2. *Permission* と *Protection Level*」の「5.2.1.2. 独自定義の *Signature Permission* で自社アプリ連携する方法」を参照すること。

4.3.2.4 結果情報を返す場合には、返送先アプリからの結果情報漏洩に注意する（必須）

query() や insert() ではリクエスト要求元アプリに結果情報として Cursor や Uri が返送される。結果情報にセンシティブな情報が含まれる場合、返送先アプリから情報漏洩する可能性がある。また update() や delete() では更新または削除されたレコード数がリクエスト要求元アプリに結果情報として返送される。まれにアプリ仕様によっては更新または削除されたレコード数がセンシティブな意味を持つ場合があるので注意すべきだ。

4.3.2.5 資産を二次的に提供する場合には、その資産の従来の保護水準を維持する（必須）

Permission により保護されている情報資産および機能資産を他のアプリに二次的に提供する場合には、提供先アプリに対して同一の Permission を要求するなどして、その保護水準を維持しなければならない。Android の Permission セキュリティモデルでは、保護された資産に対するアプリからの直接アクセスについてのみ権限管理を行う。この仕様上の特性により、アプリに取得された資産がさらに他のアプリに、保護のために必要な Permission を要求することなく提供される可能性がある。このことは Permission を再委譲していることと実質的に等価なので、Permission の再委譲問題と呼ばれる。「5.2.3.4. *Permission* の再委譲問題」を参照すること。

4.3.2.6 Content Provider の結果データの安全性を確認する（必須）

Content Provider のタイプによって若干リスクは異なるが、結果データを処理する際には、まず結果データの安全性を確認しなければならない。

利用先 Content Provider が公開 Content Provider の場合、公開 Content Provider に成り済ましたマルウェアが攻撃結果データを返送してくる可能性がある。利用先 Content Provider が非公開 Content Provider の場合、同一アプリ内から結果データを受け取るのでリスクは少ないが、結果データを無条件に安全であると考えてはならない。その他の Content Provider についても、やはり結果データの安全性を確認する必要がある。

「3.2. 入力データの安全性を確認する」を参照すること。

4.3.3 アドバンスト

4.3.3.1 コンテンツ URI の権限管理

Android 15 以前では、コンテンツ URI の権限管理に一定の制限があった。アプリ間でコンテンツを共有する際の権限チェックは主に Intent.FLAG_GRANT_READ_URI_PERMISSION などのフラグに依存し、システムレベルでの強制力や詳細なチェック機構が不足していた。また、呼び出し元アプリの特定も困難で、権限の正確な検証が難しかった。

これらの課題を解決するため、Android 15 では新たなコンテンツ URI 権限チェック API が導入され、アプリケーション間のデータ共有におけるセキュリティが大幅に強化された。

以下に、Android 15 で導入された新機能を活用したコンテンツ URI 権限管理の実装方法を示す。

コンテンツ提供アプリ

コンテンツ提供側のアプリでは、コンテンツ URI に対する権限チェックを行い、適切な権限を持つアプリからのみアクセスを許可する。

```
class MainActivity : ComponentActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        enableEdgeToEdge()
        setContent {
            MyApplicationTheme {
                Scaffold(modifier = Modifier.fillMaxSize()) { innerPadding ->
                    ContentUriPermissionDemo(
                        modifier = Modifier.padding(innerPadding)
                    )
```

(次のページに続く)

(前のページからの続き)

```
                }
            }
        }
    }
}

@Composable
fun ContentUriPermissionDemo(modifier: Modifier = Modifier) {
    val context = LocalContext.current
    var selectedUri by remember { mutableStateOf<Uri?>(null) }
    var permissionInfo by remember { mutableStateOf("権限情報はここに表示") }

    val pickImageLauncher = rememberLauncherForActivityResult(
        ActivityResultContracts.GetContent()
    ) { uri: Uri? ->
        uri?.let {
            selectedUri = it
            if (Build.VERSION.SDK_INT >= 35) {
                // Android 15 の新 API を使用して権限チェック
                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" +
                        "読み取り権限: $readPermission\n" +
                        "書き込み権限: $writePermission\n" +
                        "権限フラグ: $flags"
            } else {
                permissionInfo = "Android 15 以上が必要"
            }
        }
    }

    Column(
        modifier = modifier.fillMaxSize().padding(16.dp),
        horizontalAlignment = Alignment.CenterHorizontally,
        verticalArrangement = Arrangement.spacedBy(16.dp)
    ) {
        Button(onClick = { pickImageLauncher.launch("image/*") }) {
            Text("画像を選択")
```

(次のページに続く)

(前のページからの続き)

```
        }

        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>
```

コンテンツ利用アプリ

コンテンツ利用側のアプリでは、他アプリのコンテンツ URI にアクセスする際に適切な権限を要求し、呼び出し先アプリの要件に従ってコンテンツ URI を共有する。

```
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
```

(次のページに続く)

(前のページからの続き)

```
    val contentUri = context.intent.data

    // Composable 外で ComponentCaller 取得
    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 = "受信したコンテンツ URI",
            style = MaterialTheme.typography.headlineSmall
        )

        if (contentUri != null) {
            Text("コンテンツ URI: $contentUri")

            if (Build.VERSION.SDK_INT >= 35) {
                if (callerInfo != null) {
                    Text(
                        text = "呼び出し元情報:\n" +
                                "パッケージ名: ${callerInfo.getPackage()}\n" +
                                "ユーザー ID: ${callerInfo.getUid()}",
                        style = MaterialTheme.typography.bodyMedium
                    )
                } else {
                    Text(
                        text = "呼び出し元情報の取得に失敗しました",
                        style = MaterialTheme.typography.bodyMedium
                    )
                }
            } else {
                Text(
                    text = "ComponentCaller は Android 15 以上が必要です",
```

(次のページに続く)

(前のページからの続き)

```
                    style = MaterialTheme.typography.bodyMedium
                )
            }
        } else {
            Text(
                text = "コンテンツ URI がありません",
                style = MaterialTheme.typography.bodyMedium
            )
        }
    }
}
```

Context.checkContentUriPermissionFull API による詳細な権限チェック、requireContentUriPermissionFromCaller 属性によるシステムレベルでの権限の強制、そして ComponentCaller クラスによる呼び出し元アプリの特定機能により、より安全で透明性の高いコンテンツ共有が実現可能となった。

これらの機能を適切に活用することで、コンテンツプロバイダからの情報漏洩リスクを低減し、ユーザーのプライバシーを保護しながら、アプリ間の効率的なデータ共有を実現できる。

ただし、requireContentUriPermissionFromCaller 属性は、Android の内部実装で@hide 指定されている非公開 API となっている。そのため、将来的な仕様変更や廃止の可能性がある。現時点ではマニフェストで値を指定すれば利用できるが、今後も安定して使える保証はない

4.3.3.2 連絡先プロバイダ（CP2）の厳格化

Android 17 以上をターゲットとするアプリでは、連絡先プロバイダ（CP2）に対するアクセス制限と仕様変更が実施されている。

第一に、一時的な URI 権限（Contact Picker 経由など）を利用し、READ_CONTACTS 権限を持たずに ContactsContract.Data テーブルへアクセスする場合、厳格な SQL クエリ検証が適用される。この検証において互換性のないと判断されたクエリは拒否され、 IllegalArgumentException などの例外がスローされる仕様となっている。

第二に、プライバシー強化を目的として、個人を特定できる情報（PII）を含む機密データフィールド（ACCOUNT_NAME、ACCOUNT_TYPE、ACCOUNT_TYPE_AND_DATA_SET など）がデータビューから制限、または削除される。これらの列をクエリで直接指定すると例外が発生するか、データが取得できない。この制限に対応するためには、Data テーブルからは RAW_CONTACT_ID を取得し、それをキーとして ContactsContract.RawContacts テーブルを個別にクエリする設計へと改修する必要がある。

以下に、Android 17 の連絡先プロバイダ制限強化に対する非推奨な実装と、改修後の実装例を示す。

従来の非推奨な実装（**Android 17** では例外が発生、またはデータが取得できない)

ContactsContract.Data テーブルから直接 ACCOUNT_NAME などの機密データフィールドを取得しようとする実装である。Android 17 ではこれらの列が制限されており、厳格なクエリ検証により例外がスローされるか、データが取得できない。

```
// NG 例：機密データフィールドの直接指定によりエラーとなる
fun queryAccountNameLegacy(context: Context) {
```

(次のページに続く)

(前のページからの続き)

```
    // ※ Kotlin の仕様制約を回避するため、定数ではなく文字列リテラルを直接指定している
    val projection = arrayOf(
        ContactsContract.Data._ID,
        "account_name", // 機密データ (ContactsContract.Data.ACCOUNT_NAME)
        "account_type"  // 機密データ (ContactsContract.Data.ACCOUNT_TYPE)
    )

    try {
        val cursor = context.contentResolver.query(
            ContactsContract.Data.CONTENT_URI,
            projection,
            null,
            null,
            null
        )
        cursor?.use {
            // データの読み取り処理
        }
    } catch (e: Exception) {
        // Android 17 以降では例外が発生する
        e.printStackTrace()
    }
}
```

Android 17 対応の実装（RAW_CONTACT_ID を用いた結合・抽出)

Android 17 以降でこれらの機密データにアクセスする必要がある場合は、ContactsContract.Data テーブルからは RAW_CONTACT_ID のみを取得し、それをキーとして ContactsContract.RawContacts テーブルを個別にクエリする設計へと改修する。

```
// OK 例（Kotlin）：RAW_CONTACT_ID を利用して RawContacts テーブルから安全に抽出する
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
        )
```

(次のページに続く)

(前のページからの続き)

```
        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)
                        // 取得した機密データの利用処理
                    }
                }
            }
        }
    } catch (e: Exception) {
        e.printStackTrace()
    }
}
```

4.4 Service を作る・利用する

4.4.1 サンプルコード

Service がどのように利用されるかによって、Service が抱えるリスクや適切な防御手段が異なる。次の判定フローによって作成する Service がどのタイプであるかを判断できる。なお、作成する Service のタイプによって Service を利用する側の実装も決まるので、利用側の実装についても合わせて説明する。

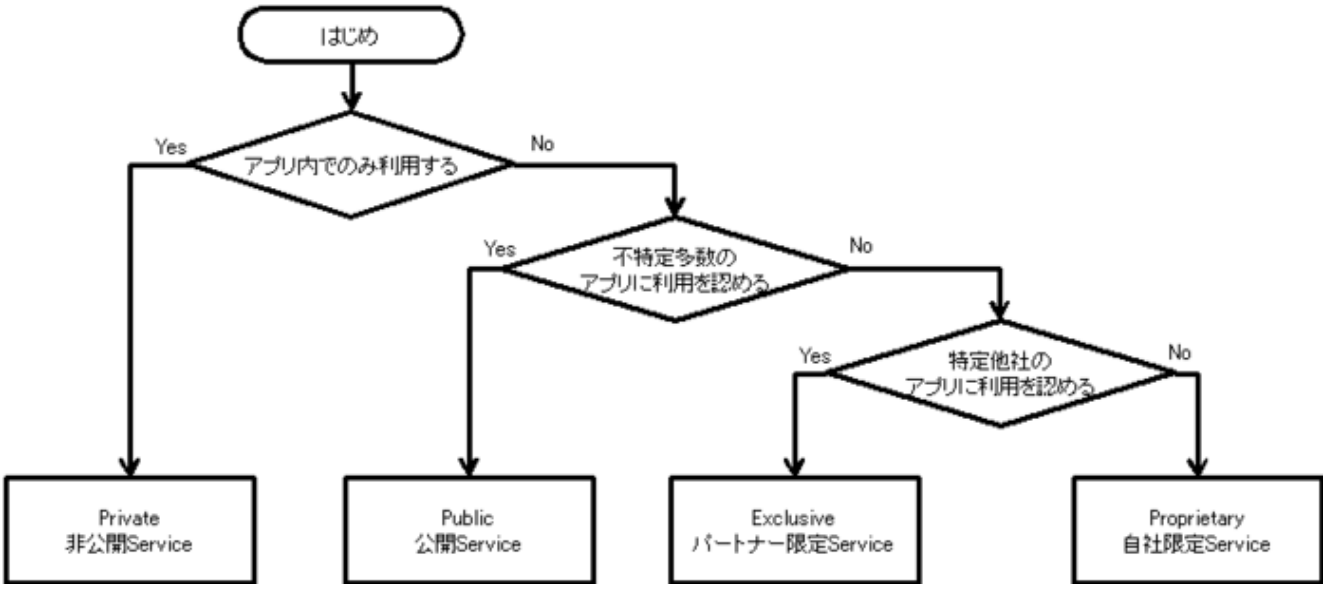

図 4.4.1　Service タイプの選択フロー

Service には複数の実装方法があり、その中から作成する Service のタイプに合った方法を選択することになる。下表の縦の項目が本文書で扱う実装方法であり、5 種類に分類した。表中の o 印は実現可能な組み合わせを示し、その他は実現不可能もしくは困難なものを示す。

なお、Service の実装方法の詳細については、「4.4.3.2. *Service* の実装方法について」および各 Service タイプのサンプルコード（表中で*印の付いたもの）を参照すること。

表 4.4.1　Service の実装方法

分類	非公開 **Service**	公開 **Service**	パートナー限定 **Service**	自社限定 **Service**
startService 型	**o***	o	-	o
IntentService 型	o	**o***	-	o
local bind 型	o	-	-	-
Messenger bind 型	o	o	-	**o***
AIDL bind 型	o	o	**o***	o

以下では 表 4.4.1 中の*印の組み合わせを使って各セキュリティタイプの Service のサンプルコードを示す。

4.4.1.1 非公開 Service を作る・利用する

非公開 Service は、同一アプリ内でのみ利用される Service であり、もっとも安全性の高い Service である。

また、非公開 Service を利用するには、クラスを指定する明示的 Intent を使えば誤って外部アプリに Intent を送信してしまうことがない。

以下、startService 型の Service を使用した例を示す。

ポイント (Service を作る)：

1. exported=”false” により、明示的に非公開設定する
2. 同一アプリからの Intent であっても、受信 Intent の安全性を確認する
3. 結果を返す場合、利用元アプリは同一アプリであるから、センシティブな情報を返送してよい

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>

    <!-- 非公開 Service -->
    <!-- ★ポイント 1★ exported="false"により、明示的に非公開設定する -->
    <service android:name=".PrivateStartService" android:exported="false"/>

    <!-- IntentService を継承した Service -->
    <!-- 非公開 Service -->
    <!-- ★ポイント 1★ exported="false"により、明示的に非公開設定する -->
    <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{
    // Service が起動するときに 1 回だけ呼び出される
    @Override
    public void onCreate() {
        Toast.makeText(this,
                       this.getClass().getSimpleName() + " - onCreate()",
                       Toast.LENGTH_SHORT).show();
    }

    // startService() が呼ばれた回数だけ呼び出される
    @Override
    public int onStartCommand(Intent intent, int flags, int startId) {
        // ★ポイント 2★ 同一アプリからの Intent であっても、受信 Intent の安全性を確認する
        // サンプルにつき割愛。「3.2 入力データの安全性を確認する」を参照。
        String param = intent.getStringExtra("PARAM");
        Toast.makeText(this,
                       String.format("パラメータ「%s」を受け取った。", param),
                       Toast.LENGTH_LONG).show();

        // サービスは明示的に終了させる
        // stopSelf や stopService を実行したときにサービスを終了する
        // START_NOT_STICKY は、メモリが少ない等で kill された場合に自動的には復帰しない
```

(次のページに続く)

(前のページからの続き)

```
        return Service.START_NOT_STICKY;
    }

    // Service が終了するときに 1 回だけ呼び出される
    @Override
    public void onDestroy() {
        Toast.makeText(this,
                       this.getClass().getSimpleName() + " - onDestroy()",
                       Toast.LENGTH_SHORT).show();
    }

    @Override
    public IBinder onBind(Intent intent) {
        // このサービスにはバインドしない
        return null;
    }
}
```

次に非公開 Service を利用する Activity のサンプルコードを示す。

ポイント (Service を利用する):

4. 同一アプリ内 Service はクラス指定の明示的 Intent で呼び出す
5. 利用先アプリは同一アプリであるから、センシティブな情報を送信してもよい
6. 結果を受け取る場合、同一アプリ内 Service からの結果情報であっても、受信データの安全性を確認する

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;
```

(次のページに続く)

(前のページからの続き)

```
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);
    }

    // サービス開始

    public void onStartServiceClick(View v) {
        // ★ポイント 4★ 同一アプリ内 Service はクラス指定の明示的 Intent で呼び出す
        Intent intent = new Intent(this, PrivateStartService.class);

        // ★ポイント 5★ 利用先アプリは同一アプリであるから、センシティブな情報を送信してもよい
        intent.putExtra("PARAM", "センシティブな情報");

        startService(intent);
    }

    // サービス停止ボタン
    public void onStopServiceClick(View v) {
        doStopService();
    }

    @Override
    public void onStop(){
        super.onStop();
        // サービスが終了していない場合は終了する
        doStopService();
    }
    // サービスを停止する
    private void doStopService() {
        // ★ポイント 4★ 同一アプリ内 Service はクラス指定の明示的 Intent で呼び出す
        Intent intent = new Intent(this, PrivateStartService.class);
        stopService(intent);
    }

    // IntentService 開始ボタン

    public void onIntentServiceClick(View v) {
```

(次のページに続く)

(前のページからの続き)

```
        // ★ポイント 4★ 同一アプリ内 Service はクラス指定の明示的 Intent で呼び出す
        Intent intent = new Intent(this, PrivateIntentService.class);

        // ★ポイント 5★ 利用先アプリは同一アプリであるから、センシティブな情報を送信してもよい
        intent.putExtra("PARAM", "センシティブな情報");

        startService(intent);
    }
}
```

4.4.1.2 公開 Service を作る・利用する

公開 Service は、不特定多数のアプリに利用されることを想定した Service である。マルウェアが送信した情報（Intent など）を受信することがあることに注意が必要である。また、Service 起動の Intent がマルウェアに受信される可能性があるため公開 Service の起動には明示的 Intent を使用し、さらに Service では<intent-filter>を宣言しないようにすべきである。

以下、IntentService 型の Service を使用した例を示す。

ポイント（Service を作る）:

1. Intent Filter を定義せず、exported=”true” を明示的に設定する
2. 受信 Intent の安全性を確認する
3. 結果を返す場合、センシティブな情報を含めない

```
AndroidManifest.xml
<?xml version="1.0" encoding="utf-8"?>
<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" >

    <!-- 最も標準的な Service -->
    <!-- ★ポイント 1★ Intent Filter を定義せず、exported="true"を明示的に設定する -->
    <service android:name=".PublicStartService" android:exported="true" />

    <!-- IntentService を継承した Service -->
    <!-- ★ポイント 1★ Intent Filter を定義せず、exported="true"を明示的に設定する -->
    <service android:name=".PublicIntentService" android:exported="true" />
```

(次のページに続く)

(前のページからの続き)

```
  </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";

    /**
     * IntentService を継承した場合、引数無しのコンストラクタを必ず用意する。
     * これが無い場合、エラーになる。
     */
    public PublicIntentService() {
        super("CreatingTypeBService");
    }
```

(次のページに続く)

(前のページからの続き)

```
    // Service が起動するときに 1 回だけ呼び出される
    @Override
    public void onCreate() {
        super.onCreate();

        Toast.makeText(this,
                       this.getClass().getSimpleName() + " - onCreate()",
                       Toast.LENGTH_SHORT).show();
    }

    // Service で行いたい処理をこのメソッドに記述する
    @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);
        }
        // ★ポイント 2★ 受信 Intent の安全性を確認する
        // 公開 Activity であるため利用元アプリがマルウェアである可能性がある。
        // サンプルにつき割愛。「3.2 入力データの安全性を確認する」を参照。
        String param = intent.getStringExtra("PARAM");
        Toast.makeText(this,
                       String.format("パラメータ「%s」を受け取った。", param),
                       Toast.LENGTH_LONG).show();
    }


    // Service が終了するときに 1 回だけ呼び出される
    @Override
    public void onDestroy() {
        Toast.makeText(this,
```

(次のページに続く)

(前のページからの続き)

```
                        this.getClass().getSimpleName() + " - onDestroy()",
                        Toast.LENGTH_SHORT).show();
    }
}
```

次に公開 Service を利用する Activity のサンプルコードを示す。

ポイント（Service を利用する）:

4. Service は明示的 Intent で呼び出す
5. センシティブな情報を送信してはならない
6. 結果を受け取る場合、結果データの安全性を確認する

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");
```

(次のページに続く)

(前のページからの続き)

```
 * 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.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 {

    // 利用先 Service 情報
    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);
    }

    // サービス開始
    public void onStartServiceClick(View v) {

        Intent intent =
            new Intent("org.jssec.android.service.publicservice.action.startservice");

        // ★ポイント 4★ Service は明示的 Intent で呼び出す
        intent.setClassName(TARGET_PACKAGE, TARGET_START_CLASS);
```

(次のページに続く)

(前のページからの続き)

```
        // ★ポイント 5★ センシティブな情報を送信してはならない
        intent.putExtra("PARAM", "センシティブではない情報");

        if (Build.VERSION.SDK_INT >= 26) {
            startForegroundService(intent);
        } else {
            startService(intent);
        }

        // ★ポイント 6★ 結果を受け取る場合、結果データの安全性を確認する
        // 本サンプルは startService() を使った Service 利用の例の為、結果情報は受け取らない
    }

    // サービス停止ボタン
    public void onStopServiceClick(View v) {
        doStopService();
    }

    // IntentService 開始ボタン

    public void onIntentServiceClick(View v) {
        Intent intent =
            new Intent("org.jssec.android.service.publicservice.action.intentservice");

        // ★ポイント 4★ Service は明示的 Intent で呼び出す
        intent.setClassName(TARGET_PACKAGE, TARGET_INTENT_CLASS);

        // ★ポイント 5★ センシティブな情報を送信してはならない
        intent.putExtra("PARAM", "センシティブではない情報");

        if (Build.VERSION.SDK_INT >= 26) {
            startForegroundService(intent);
        } else {
            startService(intent);
        }
    }

    @Override
    public void onStop(){
        super.onStop();
        // サービスが終了していない場合は終了する
        doStopService();
    }

    // サービスを停止する
```

(次のページに続く)

(前のページからの続き)

```
    private void doStopService() {
        Intent intent =
            new Intent("org.jssec.android.service.publicservice.action.startservice");

        // ★ポイント 4★ Service は明示的 Intent で呼び出す
        intent.setClassName(TARGET_PACKAGE, TARGET_START_CLASS);

        stopService(intent);
    }
}
```

4.4.1.3 パートナー限定 Service

パートナー限定 Service は、特定のアプリだけから利用できる Service である。パートナー企業のアプリと自社アプリが連携してシステムを構成し、パートナーアプリとの間で扱う情報や機能を守るために利用される。

以下、AIDL bind 型の Service を使用した例を示す。

ポイント (Service を作る) :

1. Intent Filter を定義せず、exported=”true” を明示的に設定する
2. 利用元アプリの証明書がホワイトリストに登録されていることを確認する
3. onBind(onStartCommand,onHandleIntent) で呼び出し元がパートナーかどうか判別できない
4. パートナーアプリからの Intent であっても、受信 Intent の安全性を確認する
5. パートナーアプリに開示してよい情報に限り返送してよい

なお、ホワイトリストに指定する利用先アプリの証明書ハッシュ値の確認方法は「5.2.1.3. アプリの証明書のハッシュ値を確認する方法」を参照すること。

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">

    <!-- AIDL を利用した Service -->
    <!-- ★ポイント 1★ Intent Filter を定義せず、exported="true"を明示的に設定する -->
    <service
        android:name="org.jssec.android.service.partnerservice.aidl.PartnerAIDLService"
        android:exported="true" />
```

(次のページに続く)

(前のページからの続き)

```
  </application>

</manifest>
```

今回の例では AIDL ファイルを 2 つ作成する。1 つは、Service から Activity にデータを渡すためのコールバックインターフェースで、もう 1 つは Activity から Service にデータを渡し、情報を取得するインターフェースである。なお、AIDL ファイルに記述するパッケージ名は、java ファイルに記述するパッケージ名と同様に、AIDL ファイルを作成するディレクトリ階層に一致させる必要がある。

IPartnerAIDLServiceCallback.aidl

```
package org.jssec.android.service.partnerservice.aidl;

interface IPartnerAIDLServiceCallback {
    /**
     * 値が変わった時に呼び出される
     */
    void valueChanged(String info);
}
```

IPartnerAIDLService.aidl

```
package org.jssec.android.service.partnerservice.aidl;

import org.jssec.android.service.partnerservice.aidl.IExclusiveAIDLServiceCallback;

interface IPartnerAIDLService {

    /**
     * コールバックを登録する
     */
    void registerCallback(IPartnerAIDLServiceCallback cb);

    /**
     * 情報を取得する
     */
    String getInfo(String param);

    /**
     * コールバックを解除する
     */
    void unregisterCallback(IPartnerAIDLServiceCallback cb);
}
```

PartnerAIDLService.java

```
/*
```

(次のページに続く)

(前のページからの続き)

```
 * 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;

    // Service からクライアントに通知する値
    private int mValue = 0;

    // ★ポイント 2★ 利用元アプリの証明書がホワイトリストに登録されていることを確認する
    private static PkgCertWhitelists sWhitelists = null;
    private static void buildWhitelists(Context context) {
        boolean isdebug = Utils.isDebuggable(context);
        sWhitelists = new PkgCertWhitelists();

        // パートナーアプリ org.jssec.android.service.partnerservice.aidluser の証明書ハッシュ値
を登録
        sWhitelists.add("org.jssec.android.service.partnerservice.aidluser", isdebug ?
```

(次のページに続く)

(前のページからの続き)

```
            // debug.keystore の"androiddebugkey"の証明書ハッシュ値
            "0EFB7236 328348A9 89718BAD DF57F544 D5CCB4AE B9DB34BC 1E29DD26 F77C8255" :
            // keystore の"partner key"の証明書ハッシュ値
            "1F039BB5 7861C27A 3916C778 8E78CE00 690B3974 3EB8259F E2627B8D 4C0EC35A");

        // 以下同様に他のパートナーアプリを登録...
    }

    private static boolean checkPartner(Context context, String pkgname) {
        if (sWhitelists == null) buildWhitelists(context);
        return sWhitelists.test(context, pkgname);
    }

    // コールバックを登録するオブジェクト。
    // RemoteCallbackList の提供するメソッドはスレッドセーフになっている。
    private final RemoteCallbackList<IPartnerAIDLServiceCallback> mCallbacks =
        new RemoteCallbackList<>();

    // コールバックに対して Service からデータを送信するための Handler
    protected 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;
                }
                // 通知を開始する
                // beginBroadcast() は、getBroadcastItem() で取得可能なコピーを作成している
                final int N = mCallbacks.beginBroadcast();
                for (int i = 0; i < N; i++) {
                    IPartnerAIDLServiceCallback target = mCallbacks.getBroadcastItem(i);
```

(次のページに続く)

(前のページからの続き)

```
                    try {
                        // ★ポイント 5★ パートナーアプリに開示してよい情報に限り送信してよい
                        target
                        .valueChanged("パートナーアプリに開示してよい情報 (callback from␣
↪Service) No."
                                + (++mValue));

                    } catch (RemoteException e) {
                        // RemoteCallbackList がコールバックを管理しているので、
                        // ここでは unregeister しない
                        // RemoteCallbackList.kill() によって全て解除される
                    }
                }
                // finishBroadcast() は、beginBroadcast() と対になる処理
                mCallbacks.finishBroadcast();

                // 10 秒後に繰り返す
                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);

    // AIDL で定義したインターフェース
    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
```

(次のページに続く)

(前のページからの続き)

```
                    public void run() {
                        Toast.makeText(PartnerAIDLService.this,
                                       "利用元アプリはパートナーアプリではない。",
                                        Toast.LENGTH_LONG).show();
                    }
                });
                return false;
            }
            return true;
        }
        public void registerCallback(IPartnerAIDLServiceCallback cb) {
            // ★ポイント 2★ 利用元アプリの証明書がホワイトリストに登録されていることを確認する
            if (!checkPartner()) {
                return;
            }
            if (cb != null) mCallbacks.register(cb);
        }
        public String getInfo(String param) {

            // ★ポイント 2★ 利用元アプリの証明書がホワイトリストに登録されていることを確認する
            if (!checkPartner()) {
                return null;
            }
            // ★ポイント 4★ パートナーアプリからの Intent であっても、受信 Intent の安全性を確認す
る
            // サンプルにつき割愛。「3.2 入力データの安全性を確認する」を参照。
            Message msg = new Message();
            msg.what = GETINFO_MSG;
            msg.obj =
                String.format("パートナーアプリからのメソッド呼び出し。「%s」を受信した。",
                              param);
            PartnerAIDLService.this.mHandler.sendMessage(msg);

            // ★ポイント 5★ パートナーアプリに開示してよい情報に限り返送してよい
            return "パートナーアプリに開示してよい情報 (method from Service)";
        }

        public void unregisterCallback(IPartnerAIDLServiceCallback cb) {
            // ★ポイント 2★ 利用元アプリの証明書がホワイトリストに登録されていることを確認する
            if (!checkPartner()) {
                return;
            }

            if (cb != null) mCallbacks.unregister(cb);
        }
```

(次のページに続く)

(前のページからの続き)

```
        };

    @Override
    public IBinder onBind(Intent intent) {
        // ★ポイント 3★ onBind(onStartCommand,onHandleIntent) で呼び出し元がパートナーかどうか判
別できない
        // AIDL で定義したメソッドの呼び出し毎にチェックが必要になる。

        return mBinder;
    }

    @Override
    public void onCreate() {
        Toast.makeText(this,
                       this.getClass().getSimpleName() + " - onCreate()",
                       Toast.LENGTH_SHORT).show();

        // Service が実行中の間は、定期的にインクリメントした数字を通知する
        mHandler.sendEmptyMessage(REPORT_MSG);

    }

    @Override
    public void onDestroy() {
        Toast.makeText(this,
                      this.getClass().getSimpleName() + " - onDestroy()",
                       Toast.LENGTH_SHORT).show();

        // コールバックを全て解除する
        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
 *
```

(次のページに続く)

(前のページからの続き)

```
 *      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 は 32 バイト
        sha256 = sha256.toUpperCase();
        if (sha256.replaceAll("[0-9A-F]+", "").length() != 0)
            return false;    // 0-9A-F 以外の文字がある

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

    public boolean test(Context ctx, String pkgname) {
        // pkgname に対応する正解のハッシュ値を取得する
        String correctHash = mWhitelists.get(pkgname);

        // pkgname の実際のハッシュ値と正解のハッシュ値を比較する
        if (Build.VERSION.SDK_INT >= 28) {
            // ★ API Level >= 28 では Package Manager の API で直接検証が可能
            PackageManager pm = ctx.getPackageManager();
```

(次のページに続く)

(前のページからの続き)

```
            return pm.hasSigningCertificate(pkgname,
                                            Utils.hex2Bytes(correctHash),
                                            CERT_INPUT_SHA256);
        } else {
            // API Level < 28 の場合は PkgCert の機能を利用する
            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));
```

(次のページに続く)

(前のページからの続き)

```
    }

    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);
            if (pkginfo.signatures.length != 1) return null;  // 複数署名は扱わない
            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();
    }
}
```

次にパートナー限定 Service を利用する Activity のサンプルコードを示す。

ポイント (Service を利用する) :

6. 利用先パートナー限定 Service アプリの証明書がホワイトリストに登録されていることを確認する
7. 利用先パートナー限定アプリに開示してよい情報に限り送信してよい
8. 明示的 Intent によりパートナー限定 Service を呼び出す
9. パートナー限定アプリからの結果情報であっても、受信 Intent の安全性を確認する

PartnerAIDLUserActivity.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.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;

    // ★ポイント 6★ 利用先パートナー限定 Service アプリの証明書がホワイトリストに登録されていることを確認する
```

(次のページに続く)

(前のページからの続き)

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

        // パートナー限定 Service アプリ org.jssec.android.service.partnerservice.aidl の証明書
ハッシュ値を登録
        sWhitelists.add("org.jssec.android.service.partnerservice.aidl", isdebug ?
            // debug.keystore の"androiddebugkey"の証明書ハッシュ値
            "0EFB7236 328348A9 89718BAD DF57F544 D5CCB4AE B9DB34BC 1E29DD26 F77C8255" :
            // keystore の"my company key"の証明書ハッシュ値
            "D397D343 A5CBC10F 4EDDEB7C A10062DE 5690984F 1FB9E88B D7B3A7C2 42E142CA");

        // 以下同様に他のパートナー限定 Service アプリを登録...
    }
    private static boolean checkPartner(Context context, String pkgname) {
        if (sWhitelists == null) buildWhitelists(context);
        return sWhitelists.test(context, pkgname);
    }

    // 利用先のパートナー限定 Activity に関する情報
    private static final String TARGET_PACKAGE =
        "org.jssec.android.service.partnerservice.aidl";
    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("コールバックで「%s」を受信した。", info),
                               Toast.LENGTH_SHORT).show();
                break;
            }
            default:
                super.handleMessage(msg);
                break;
```

(次のページに続く)

(前のページからの続き)

```
            } // switch
        }

    }

    private final ReceiveHandler mHandler = new ReceiveHandler(this);

    // AIDL で定義したインターフェース。Service からの通知を受け取る。
    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);
            }
        };

    // AIDL で定義したインターフェース。Service へ通知する。
    private IPartnerAIDLService mService = null;

    // Service と接続する時に利用するコネクション。bindService で実装する場合は必要になる。
    private ServiceConnection mConnection = new ServiceConnection() {

        // Service に接続された場合に呼ばれる
        @Override
        public void onServiceConnected(ComponentName className, IBinder service) {
            mService = IPartnerAIDLService.Stub.asInterface(service);

            try{
                // Service に接続
                mService.registerCallback(mCallback);

            }catch(RemoteException e){
                // Service が異常終了した場合
            }

            Toast.makeText(mContext,
                           "Connected to service", Toast.LENGTH_SHORT).show();
        }

        // Service が異常終了して、コネクションが切断された場合に呼ばれる
        @Override
        public void onServiceDisconnected(ComponentName className) {
            Toast.makeText(mContext,
                           "Disconnected from service", Toast.LENGTH_SHORT).show();
```

(次のページに続く)

(前のページからの続き)

```
        }
    };

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

        setContentView(R.layout.partnerservice_activity);

        mContext = this;
    }

    // サービス開始ボタン
    public void onStartServiceClick(View v) {
        // bindService を実行する
        doBindService();
    }

    // 情報取得ボタン
    public void onGetInfoClick(View v) {
        getServiceinfo();
    }

    // サービス停止ボタン
    public void onStopServiceClick(View v) {
        doUnbindService();
    }

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

    /**
     * Service に接続する
     */
    private void doBindService() {
        if (!mIsBound){
            // ★ポイント 6★ 利用先パートナー限定 Service アプリの証明書がホワイトリストに登録されていることを確認する
            if (!checkPartner(this, TARGET_PACKAGE)) {
                Toast.makeText(this,
                               "利用先 Service アプリはホワイトリストに登録されていない。",
                               Toast.LENGTH_LONG).show();
```

(次のページに続く)

(前のページからの続き)

```
            return;
        }

        Intent intent = new Intent();

        // ★ポイント 7★ 利用先パートナー限定アプリに開示してよい情報に限り送信してよい
        intent.putExtra("PARAM", "パートナーアプリに開示してよい情報");

        // ★ポイント 8★ 明示的 Intent によりパートナー限定 Service を呼び出す
        intent.setClassName(TARGET_PACKAGE, TARGET_CLASS);

        bindService(intent, mConnection, Context.BIND_AUTO_CREATE);
        mIsBound = true;
    }
}

/**
 * Service への接続を切断する
 */
private void doUnbindService() {
    if (mIsBound) {
        // 登録していたレジスタがある場合は解除
        if(mService != null){
            try{
                mService.unregisterCallback(mCallback);
            }catch(RemoteException e){
                // Service が異常終了していた場合
                // サンプルにつき処理は割愛
            }
        }

        unbindService(mConnection);

        Intent intent = new Intent();

        // ★ポイント 8★ 明示的 Intent によりパートナー限定 Service を呼び出す
        intent.setClassName(TARGET_PACKAGE, TARGET_CLASS);

        stopService(intent);

        mIsBound = false;
    }
}

/**
```

(次のページに続く)

(前のページからの続き)

```
     * Service から情報を取得する
     */
    void getServiceinfo() {
        if (mIsBound && mService != null) {
            String info = null;

            try {
                // ★ポイント 7★ 利用先パートナー限定アプリに開示してよい情報に限り送信してよい
                info =
                    mService.getInfo("パートナーアプリに開示してよい情報 (method from activity)
↪");
            } catch (RemoteException e) {
                e.printStackTrace();
            }
            // ★ポイント 9★ パートナー限定アプリからの結果情報であっても、受信 Intent の安全性を確認す
る
            // サンプルにつき割愛。「3.2 入力データの安全性を確認する」を参照。
            Toast.makeText(mContext,
                           String.format("サービスから「%s」を取得した。", 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;
```

(次のページに続く)

(前のページからの続き)

```
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 は 32 バイト
        sha256 = sha256.toUpperCase();
        if (sha256.replaceAll("[0-9A-F]+", "").length() != 0)
            return false;    // 0-9A-F 以外の文字がある

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

    public boolean test(Context ctx, String pkgname) {
        // pkgname に対応する正解のハッシュ値を取得する
        String correctHash = mWhitelists.get(pkgname);

        // pkgname の実際のハッシュ値と正解のハッシュ値を比較する
        if (Build.VERSION.SDK_INT >= 28) {
            // ★ API Level >= 28 では Package Manager の API で直接検証が可能
            PackageManager pm = ctx.getPackageManager();
            return pm.hasSigningCertificate(pkgname,
                                            Utils.hex2Bytes(correctHash),
                                            CERT_INPUT_SHA256);
        } else {
            // API Level < 28 の場合は PkgCert の機能を利用する
            return PkgCert.test(ctx, pkgname, correctHash);
        }
    }
}
```

PkgCert.java

(次のページに続く)

(前のページからの続き)

```
/*
 * 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);
            if (pkginfo.signatures.length != 1) return null;  // 複数署名は扱わない
            Signature sig = pkginfo.signatures[0];
            byte[] cert = sig.toByteArray();
            byte[] sha256 = computeSha256(cert);
```

(次のページに続く)

(前のページからの続き)

```
            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 自社限定 Service

自社限定 Service は、自社以外のアプリから利用されることを禁止する Service である。複数の自社製アプリでシステムを構成し、自社アプリが扱う情報や機能を守るために利用される。

以下、Messenger bind 型の Service を使用した例を示す。

ポイント（Service を作る）:

1. 独自定義 Signature Permission を定義する
2. 独自定義 Signature Permission を要求宣言する
3. Intent Filter を定義せず、exported=”true” を明示的に設定する
4. 独自定義 Signature Permission が自社アプリにより定義されていることを確認する
5. 自社アプリからの Intent であっても、受信 Intent の安全性を確認する
6. 利用元アプリは自社アプリであるから、センシティブな情報を返送してよい
7. APK を Export するときに、利用元アプリと同じ開発者鍵で APK を署名する

AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
```

(次のページに続く)

(前のページからの続き)

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
          >

  <!-- ★ポイント 1 ★ 独自定義 Signature Permission を定義する -->
  <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" >

    <!-- Messenger を利用した Service -->
    <!-- ★ポイント 2 ★ 独自定義 Signature Permission を要求宣言する -->
    <!-- ★ポイント 3 ★ Intent Filter を定義せず、exported="true"を明示的に設定する -->
    <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;
```

(次のページに続く)

(前のページからの続き)

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

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{
    // 自社の Signature Permission
    private static final String MY_PERMISSION =
        "org.jssec.android.service.inhouseservice.messenger.MY_PERMISSION";

    // 自社の証明書のハッシュ値
    private static String sMyCertHash = null;
    private static String myCertHash(Context context) {
        if (sMyCertHash == null) {
            if (Utils.isDebuggable(context)) {
                // debug.keystore の"androiddebugkey"の証明書ハッシュ値
                sMyCertHash = "0EFB7236 328348A9 89718BAD DF57F544 D5CCB4AE B9DB34BC␣
↪1E29DD26 F77C8255";
            } else {
                // keystore の"my company key"の証明書ハッシュ値
                sMyCertHash = "D397D343 A5CBC10F 4EDDEB7C A10062DE 5690984F 1FB9E88B␣
↪D7B3A7C2 42E142CA";
            }
        }
        return sMyCertHash;
    }


    // Service のクライアント（データ送信先）をリストで管理する
    private ArrayList<Messenger> mClients = new ArrayList<Messenger>();

    // クライアントからのデータを受信するときに利用する Messenger
```

(次のページに続く)

(前のページからの続き)

```
    private final Messenger mMessenger = new Messenger(new ServiceSideHandler(mClients));

    // クライアントから受け取った Message を処理する Handler
    private static class ServiceSideHandler extends Handler{

        private ArrayList<Messenger> mClients;

        public ServiceSideHandler(ArrayList<Messenger> clients){
            this.mClients = clients;
        }

        @Override
        public void handleMessage(Message msg){
            switch(msg.what){
            case CommonValue.MSG_REGISTER_CLIENT:
                // クライアントから受け取った Messenger を追加
                mClients.add(msg.replyTo);
                break;
            case CommonValue.MSG_UNREGISTER_CLIENT:
                mClients.remove(msg.replyTo);
                break;
            case CommonValue.MSG_SET_VALUE:
                // クライアントにデータを送る
                sendMessageToClients(mClients);
                break;
            default:
                super.handleMessage(msg);
                break;
            }
        }
    }

    /**
     * クライアントにデータを送る
     */
    private static void sendMessageToClients(ArrayList<Messenger> mClients){

        // ★ポイント 6★ 利用元アプリは自社アプリであるから、センシティブな情報を返送してよい
        String sendValue = "センシティブな情報 (from Service)";

        // 登録されているクライアントへ、順番に送信する
        // ループ途中で remove しても全てのデータにアクセスしたいので Iterator を利用する
        Iterator<Messenger> ite = mClients.iterator();
        while(ite.hasNext()){
            try {
```

(次のページに続く)

(前のページからの続き)

```
                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) {
                // クライアントが存在しない場合は、リストから取り除く
                ite.remove();
            }
        }
    }

    @Override
    public IBinder onBind(Intent intent) {

        // ★ポイント 4 ★ 独自定義 Signature Permission が自社アプリにより定義されていることを確認する
        if (!SigPerm.test(this, MY_PERMISSION, myCertHash(this))) {
            Toast.makeText(this,
                           "独自定義 Signature Permission が自社アプリにより定義されていない。",
                           Toast.LENGTH_LONG).show();
            return null;
        }

        // ★ポイント 5 ★ 自社アプリからの Intent であっても、受信 Intent の安全性を確認する
        // サンプルにつき割愛。「3.2 入力データの安全性を確認する」を参照。
        String param = intent.getStringExtra("PARAM");
        Toast.makeText(this,
                       String.format("パラメータ「%s」を受け取った。", 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();
```

(次のページに続く)

(前のページからの続き)

```
    }
}
```

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{
            // sigPermName を定義したアプリのパッケージ名を取得する
            PackageManager pm = ctx.getPackageManager();
            PermissionInfo pi =
                pm.getPermissionInfo(sigPermName, PackageManager.GET_META_DATA);
            String pkgname = pi.packageName;
            // 非 Signature Permission の場合は失敗扱い
            if (pi.protectionLevel != PermissionInfo.PROTECTION_SIGNATURE) return false;
            // pkgname の実際のハッシュ値と正解のハッシュ値を比較する
```

(次のページに続く)

(前のページからの続き)

```
            if (Build.VERSION.SDK_INT >= 28) {
                // ★ API Level >= 28 では Package Manager の API で直接検証が可能
                return pm.hasSigningCertificate(pkgname,
                                                Utils.hex2Bytes(correctHash),
                                                CERT_INPUT_SHA256);
            } else {
                // API Level < 28 の場合は PkgCert を利用し、ハッシュ値を取得して比較する
                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 {
```

(次のページに続く)

(前のページからの続き)

```
    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);
            if (pkginfo.signatures.length != 1) return null;  // 複数署名は扱わない
            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();
    }
}
```

★ポイント 7 ★ APK を Export するときに、利用元アプリと同じ開発者鍵で APK を署名する。

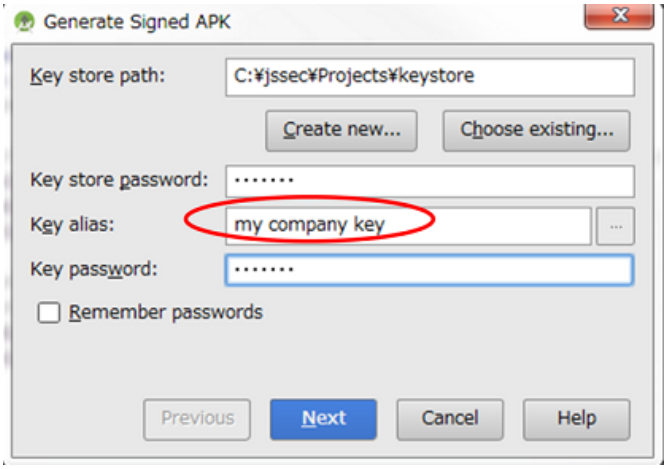

図 4.4.2　利用元アプリと同じ開発者鍵で APK を署名する

次に自社限定 Service を利用する Activity のサンプルコードを示す。

ポイント (Service を利用する) : 8. 独自定義 Signature Permission を利用宣言する 9. 独自定義 Signature Permission が自社アプリにより定義されていることを確認する 10. 利用先アプリの証明書が自社の証明書であることを確認する 11. 利用先アプリは自社アプリであるから、センシティブな情報を送信してもよい 12. 明示的 Intent により自社限定 Service を呼び出す 13. 自社アプリからの結果情報であっても、受信 Intent の安全性を確認する 14. APK を Export するときに、利用先アプリと同じ開発者鍵で APK を署名する

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>

  <!-- ★ポイント 8★ 独自定義 Signature Permission を利用宣言する -->
  <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" />
```

(次のページに続く)

(前のページからの続き)

```
      </intent-filter>
    </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 {
```

(次のページに続く)

(前のページからの続き)

```
    private boolean mIsBound;
    private Context mContext;

    // 利用先の Activity 情報
    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";

    // 自社の Signature Permission
    private static final String MY_PERMISSION =
        "org.jssec.android.service.inhouseservice.messenger.MY_PERMISSION";

    // 自社の証明書のハッシュ値
    private static String sMyCertHash = null;
    private static String myCertHash(Context context) {
        if (sMyCertHash == null) {
            if (Utils.isDebuggable(context)) {
                // debug.keystore の"androiddebugkey"の証明書ハッシュ値
                sMyCertHash = "0EFB7236 328348A9 89718BAD DF57F544 D5CCB4AE B9DB34BC␣
↪1E29DD26 F77C8255";
            } else {
                // keystore の"my company key"の証明書ハッシュ値
                sMyCertHash = "D397D343 A5CBC10F 4EDDEB7C A10062DE 5690984F 1FB9E88B␣
↪D7B3A7C2 42E142CA";
            }
        }
        return sMyCertHash;
    }

    // Service からデータを受信するときに利用する Messenger
    private Messenger mServiceMessenger = null;

    // Service にデータを送信するときに利用する Messenger
    private final Messenger mActivityMessenger = new Messenger(new ActivitySideHandler());

    // Service から受け取った Message を処理する Handler
    private class ActivitySideHandler extends Handler {

        @Override
        public void handleMessage(Message msg) {
            switch (msg.what) {
            case CommonValue.MSG_SET_VALUE:
                Bundle data = msg.getData();
```

(次のページに続く)

(前のページからの続き)

```
                String info = data.getString("key");
                // ★ポイント 13★ 自社アプリからの結果情報であっても、値の安全性を確認する
                // サンプルにつき割愛。「3.2 入力データの安全性を確認する」を参照。
                Toast.makeText(mContext, String.format("サービスから「%s」を取得した。", info),
                               Toast.LENGTH_SHORT).show();
                break;
            default:
                super.handleMessage(msg);
            }
        }
    }

    // Service と接続する時に利用するコネクション。bindService で実装する場合は必要になる。
    private ServiceConnection mConnection = new ServiceConnection() {

        // Service に接続された場合に呼ばれる
        @Override
        public void onServiceConnected(ComponentName className, IBinder service) {
            mServiceMessenger = new Messenger(service);
            Toast.makeText(mContext, "Connect to service", Toast.LENGTH_SHORT).show();

            try {
                // Service に自分の Messenger を渡す
                Message msg = Message.obtain(null, CommonValue.MSG_REGISTER_CLIENT);
                msg.replyTo = mActivityMessenger;
                mServiceMessenger.send(msg);
            } catch (RemoteException e) {
                // Service が異常終了していた場合
            }
        }

        // Service が異常終了して、コネクションが切断された場合に呼ばれる
        @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.publicservice_activity);
```

(次のページに続く)

(前のページからの続き)

```
        mContext = this;
    }
    // サービス開始ボタン
    public void onStartServiceClick(View v) {
        // bindService を実行する
        doBindService();
    }

    // 情報取得ボタン
    public void onGetInfoClick(View v) {
        getServiceinfo();
    }

    // サービス停止ボタン
    public void onStopServiceClick(View v) {
        doUnbindService();
    }

    @Override
    protected void onDestroy() {
        super.onDestroy();
        doUnbindService();
    }

    /**
     * Service に接続する
     */
    void doBindService() {
        if (!mIsBound){
            // ★ポイント 9★ 独自定義 Signature Permission が自社アプリにより定義されていることを確認
する
            if (!SigPerm.test(this, MY_PERMISSION, myCertHash(this))) {
                Toast.makeText(this,
                               "独自定義 Signature Permission が自社アプリにより定義されていない。",
                               Toast.LENGTH_LONG).show();
                return;
            }

            // ★ポイント 10★ 利用先アプリの証明書が自社の証明書であることを確認する
            if (!PkgCert.test(this, TARGET_PACKAGE, myCertHash(this))) {
                Toast.makeText(this,
                               "利用先サービスは自社アプリではない。",
                               Toast.LENGTH_LONG).show();
                return;
```

(次のページに続く)

(前のページからの続き)

```
            }

            Intent intent = new Intent();

            // ★ポイント 11 ★ 利用先アプリは自社アプリであるから、センシティブな情報を送信してもよい
            intent.putExtra("PARAM", "センシティブな情報");

            // ★ポイント 12 ★ 明示的 Intent により自社限定 Service を呼び出す
            intent.setClassName(TARGET_PACKAGE, TARGET_CLASS);

            bindService(intent, mConnection, Context.BIND_AUTO_CREATE);
            mIsBound = true;
        }
    }

    /**
     * Service への接続を切断する
     */
    void doUnbindService() {
        if (mIsBound) {
            unbindService(mConnection);
            mIsBound = false;
        }
    }

    /**
     * Service から情報を取得する
     */
    void getServiceinfo() {
        if (mServiceMessenger != null) {
            try {
                // 情報の送信を要求する
                Message msg = Message.obtain(null, CommonValue.MSG_SET_VALUE);
                mServiceMessenger.send(msg);
            } catch (RemoteException e) {
                // Service が異常終了していた場合
            }
        }
    }
}
```

SigPerm.java

```
/*
 * Copyright (C) 2012-2026 Japan Smartphone Security Association
```

(次のページに続く)

(前のページからの続き)

```
 *
 * 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{
            // sigPermName を定義したアプリのパッケージ名を取得する
            PackageManager pm = ctx.getPackageManager();
            PermissionInfo pi =
                pm.getPermissionInfo(sigPermName, PackageManager.GET_META_DATA);
            String pkgname = pi.packageName;
            // 非 Signature Permission の場合は失敗扱い
            if (pi.protectionLevel != PermissionInfo.PROTECTION_SIGNATURE) return false;
            // pkgname の実際のハッシュ値と正解のハッシュ値を比較する
            if (Build.VERSION.SDK_INT >= 28) {
                // ★ API Level >= 28 では Package Manager の API で直接検証が可能
                return pm.hasSigningCertificate(pkgname,
                                                Utils.hex2Bytes(correctHash),
                                                CERT_INPUT_SHA256);
            } else {
                // API Level < 28 の場合は PkgCert を利用し、ハッシュ値を取得して比較する
```

(次のページに続く)

(前のページからの続き)

```
                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));
    }
```

(次のページに続く)

(前のページからの続き)

```
    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);
            if (pkginfo.signatures.length != 1) return null;  // 複数署名は扱わない
            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();
    }
}
```

★ポイント 14 ★ APK を Export するときに、利用先アプリと同じ開発者鍵で APK を署名する。

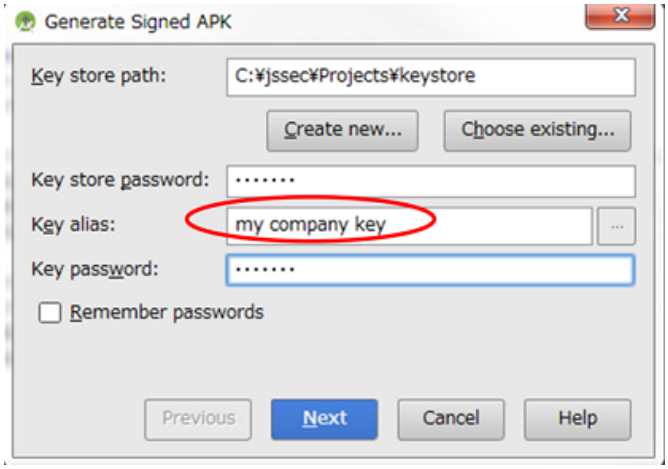

図 4.4.3　利用先アプリと同じ開発者鍵で APK を署名する

4.4.2 ルールブック

Service 実装時には以下のルールを守ること。

1. アプリ内でのみ使用する *Service* は非公開設定する（必須）
2. 受信データの安全性を確認する（必須）
3. 独自定義 *Signature Permission* は、自社アプリが定義したことを確認して利用する（必須）
4. 連携するタイミングで *Service* の機能を提供するかを判定する（必須）
5. 結果情報を返す場合には、返送先アプリからの結果情報漏洩に注意する（必須）
6. 利用先 *Service* が固定できる場合は明示的 *Intent* で *Service* を利用する（必須）
7. 他社の特定アプリと連携する場合は利用先 *Service* を確認する（必須）
8. 資産を二次的に提供する場合には、その資産の従来の保護水準を維持する（必須）
9. センシティブな情報はできる限り送らない（推奨）

4.4.2.1 アプリ内でのみ使用する **Service** は非公開設定する（必須）

アプリ内（または、同じ UID）でのみ使用される Service は非公開設定する。これにより、他のアプリから意図せず Intent を受け取ってしまうことがなくなり、アプリの機能を利用される、アプリの動作に異常をきたす等の被害を防ぐことができる。

実装上は AndroidManifest.xml で Service を定義する際に、exported 属性を false にするだけである。

```
AndroidManifest.xml
    <!-- 非公開 Service -->
    <!-- ★ 4.4.1.1 - ポイント 1 ★ exported="false"により、明示的に非公開設定する -->
    <service android:name=".PrivateStartService" android:exported="false"/>
```

また、ケースは少ないと思われるが、同一アプリ内からのみ利用される Service であり、かつ Intent Filter を設置するような設計はしてはならない。Intent Filter の性質上、同一アプリ内の非公開 Service を呼び出すつもりでも、Intent Filter 経由で呼び出したときに意図せず他アプリの公開 Service を呼び出してしまう場合が存在するからである。

AndroidManifest.xml(非推奨)

```
    <!-- 非公開 Service -->
    <!-- ★ 4.4.1.1 - ポイント 1 ★ exported="false"により、明示的に非公開設定する -->
    <service android:name=".PrivateStartService" android:exported="false">
        <intent-filter>
            <action android:name=" org.jssec.android.service.OPEN />
        </intent-filter>
    </service>
```

「4.4.3.1. *exported* 設定と *intent-filter* 設定の組み合わせ *(Service* の場合*)*」も参照すること。

4.4.2.2 受信データの安全性を確認する（必須）

Service も Activity と同様に、受信 Intent のデータを処理する際には、まず受信 Intent の安全性を確認しなければならない。Service を利用する側も Service からの結果 (として受信した）情報の安全性を確認する必要がある。Activity の「4.1.2.5. 受信 *Intent* の安全性を確認する（必須）」「4.1.2.9. 利用先 *Activity* からの戻り *Intent* の安全性を確認する（必須）」も参照すること。

Service においては、Intent 以外にもメソッドの呼び出しや Message によるデータの送受信などがあるため、それぞれ注意して実装を行わなければならない。

「3.2. 入力データの安全性を確認する」を参照すること。

4.4.2.3 独自定義 **Signature Permission** は、自社アプリが定義したことを確認して利用する（必須）

自社アプリだけから利用できる自社限定 Service を作る場合、独自定義 Signature Permission により保護しなければならない。AndroidManifest.xml での Permission 定義、Permission 要求宣言だけでは保護が不十分であるため、「5.2. *Permission* と *Protection Level*」の「5.2.1.2. 独自定義の *Signature Permission* で自社アプリ連携する方法」を参照すること。

4.4.2.4 連携するタイミングで **Service** の機能を提供するかを判定する（必須）

Intent パラメータの確認や独自定義 Signature Permission の確認といったセキュリティチェックを onCreate に入れてはいけない。その理由は、Service が起動中に新しい要求を受けたときに onCreate の処理が実施されないためである。したがって、startService によって開始される Service を実装する場合は、onStartCommand（IntentService を利用する場合は onHandleIntent）で判定を行わなければならない。bindService で開始する Service を実装する場合も同様のことが言えるので、onBind で判定をしなければならない。

4.4.2.5 結果情報を返す場合には、返送先アプリからの結果情報漏洩に注意する（必須）

Service のタイプによって結果情報の返送先（コールバックの呼び出し先や Message の送信先）アプリの信用度が異なる。返送先がマルウェアである可能性も考慮して十分に情報漏洩に対する配慮をしなければならない。

詳細は、Activity の「4.1.2.7. 結果情報を返す場合には、返送先アプリからの結果情報漏洩に注意する（必須）」を参照すること。

4.4.2.6 利用先 **Service** が固定できる場合は明示的 **Intent** で **Service** を利用する（必須）

暗黙的 Intent により Service を利用すると、Intent Filter の定義が同じ場合には先にインストールした Service に Intent が送信されてしまう。もし意図的に同じ Intent Filter を定義したマルウェアが先にインストールされていた場合、マルウェアに Intent が送信されてしまい、情報漏洩が生じる。一方、明示的 Intent により Service を利用すると、指定した Service 以外が Intent を受信することはなく比較的安全である。

ただし、別途考慮すべき点があるので、Activity の「4.1.2.8. 利用先 *Activity* が固定できる場合は明示的 *Intent* で *Activity* を利用する（必須）」を参照すること。

4.4.2.7 他社の特定アプリと連携する場合は利用先 **Service** を確認する（必須）

他社の特定アプリと連携する場合にはホワイトリストによる確認方法がある。自アプリ内に利用先アプリの証明書ハッシュを予め保持しておく。利用先の証明書ハッシュと保持している証明書ハッシュが一致するかを確認することで、なりすましアプリに Intent を発行することを防ぐことができる。具体的な実装方法についてはサンプルコードセクション「4.4.1.3. パートナー限定 *Service*」を参照すること。

4.4.2.8 資産を二次的に提供する場合には、その資産の従来の保護水準を維持する（必須）

Permission により保護されている情報資産および機能資産を他のアプリに二次的に提供する場合には、提供先アプリに対して同一の Permission を要求するなどして、その保護水準を維持しなければならない。Android の Permission セキュリティモデルでは、保護された資産に対するアプリからの直接アクセスについてのみ権限管理を行う。この仕様上の特性により、アプリに取得された資産がさらに他のアプリに、保護のために必要な Permission を要求することなく提供される可能性がある。このことは Permission を再委譲していることと実質的に等価なので、Permission の再委譲問題と呼ばれる。「5.2.3.4. *Permission* の再委譲問題」を参照すること。

4.4.2.9 センシティブな情報はできる限り送らない （推奨）

不特定多数のアプリと連携する場合にはセンシティブな情報を送ってはならない。

センシティブな情報を Service と受け渡しする場合、その情報の漏洩リスクを検討しなければならない。公開 Service に送付した情報は必ず漏洩すると考えなければならない。またパートナー限定 Service や自社限定 Service に送付した情報もそれら Service の実装に依存して情報漏洩リスクの大小がある。

センシティブな情報はできるだけ送付しないように工夫すべきである。送付する場合も、利用先 Service は信頼できる Service に限定し、情報が LogCat などに漏洩しないように配慮しなければならない。

4.4.3 アドバンスト

4.4.3.1 **exported** 設定と **intent-filter** 設定の組み合わせ **(Service** の場合**)**

このガイド文書では、Service の用途から非公開 Service、公開 Service、パートナー限定 Service、自社限定 Service の 4 タイプの Service について実装方法を述べている。各タイプに許されている AndroidManifest.xml の exported 属性と intent-filter 要素の組み合わせを次の表にまとめた。作ろうとしている Service のタイプと exported 属性および intent-filter 要素の対応が正しいことを確認すること。

表 4.4.2　exported 設定と intent-filter 設定の組み合わせ（Service の場合）

exported 属性の値	intent-filter 定義がある	intent-filter 定義がない
true	（使用禁止）	公開、パートナー限定、自社限定
false	（使用禁止）	非公開
無指定	（使用禁止）	（使用禁止）

Service の exported 属性が無指定である場合にその Service が公開されるか非公開となるかは、intent-filter の定義の有無により決まるが[*15]、本ガイドでは Service の exported 属性を「無指定」にすることを禁止している。前述のような API のデフォルトの挙動に頼る実装をすることは避けるべきであり、exported 属性のようなセキュリティ上重要な設定を明示的に有効化する手段があるのであればそれを利用すべきであると考えられるためである。

「表 4.4.2 *exported* 設定と *intent-filter* 設定の組み合わせ（*Service* の場合）」において「intent-filter 定義がある」をすべて「(使用禁止)」にしているのは、暗黙的インテントを使って Service を開始すると、どの Service がインテントに応答するかを把握できず、悪意ある Service が応答してしまう可能性があるためである。また、「intent-filter 定義がある」&「exported=”false”」を使用禁止にしているのは、Android の振る舞いとして、同一アプリ内の非公開 Service を呼び出したつもりでも、意図せず他アプリの公開 Service を呼び出してしまう場合が存在するためである。

具体的には、Android は以下のような振る舞いをするのでアプリ設計時に検討が必要である。

- 複数の Service で同じ内容の intent-filter を定義した場合、先にインストールしたアプリ内の Service の定義が優先される
- 暗黙的 Intent を使った場合は、OS によって優先の Service が自動的に選ばれて、呼び出される。

以下の 3 つの図で Android の振る舞いによる意図せぬ呼び出しが起こる仕組みを説明する。図 4.4.4 は、同一アプリ内からしか非公開 Service(アプリ A) を暗黙的 Intent で呼び出せない正常な動作の例である。Intent-filter(図中 action=”X”) を定義しているのが、アプリ A しかいないので意図通りの動きとなっている。

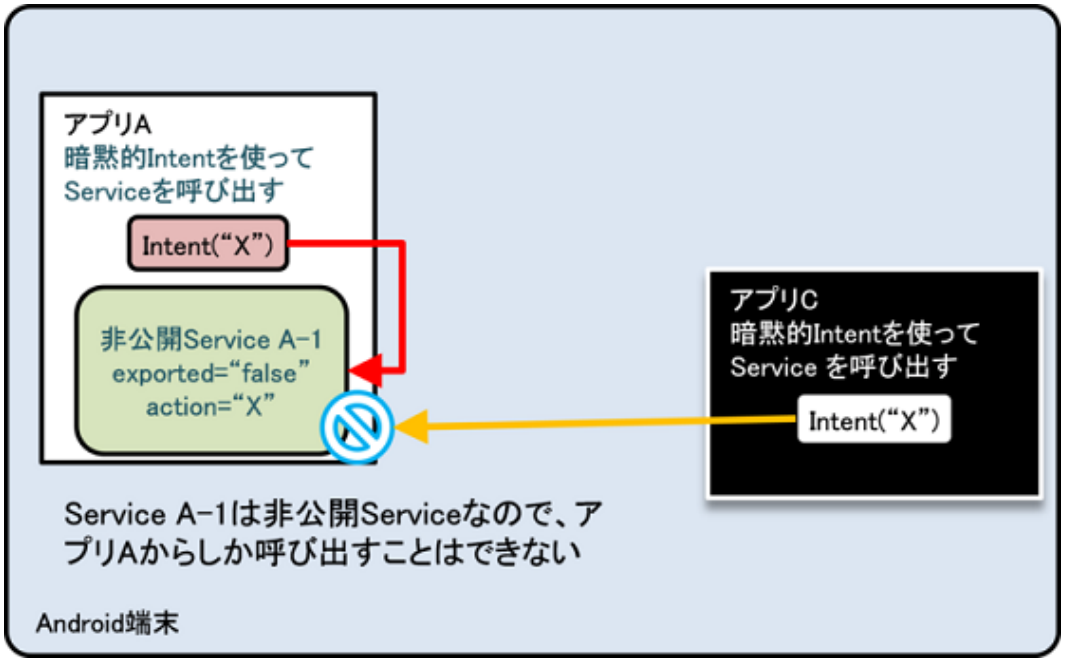

図 4.4.4　正常な動作の例

[*15] intent-filter が定義されていれば公開 Service、定義されていなければ非公開 Service となる。https://developer.android.com/guide/topics/manifest/service-element#exported

図 4.4.5 および 図 4.4.6 は、アプリ A に加えてアプリ B でも同じ intent-filter(図中 action=”X”) を定義している場合である。

図 4.4.5 は、アプリ A →アプリ B の順でインストールされた場合である。この場合、アプリ C が暗黙的 Intent を送信すると、非公開の Service(A-1) を呼び出そうとして失敗する。一方、アプリ A は暗黙的 Intent を使って意図通りに同一アプリ内の非公開 Service を呼び出せるので、セキュリティの (マルウェア対策の) 面では問題は起こらない。

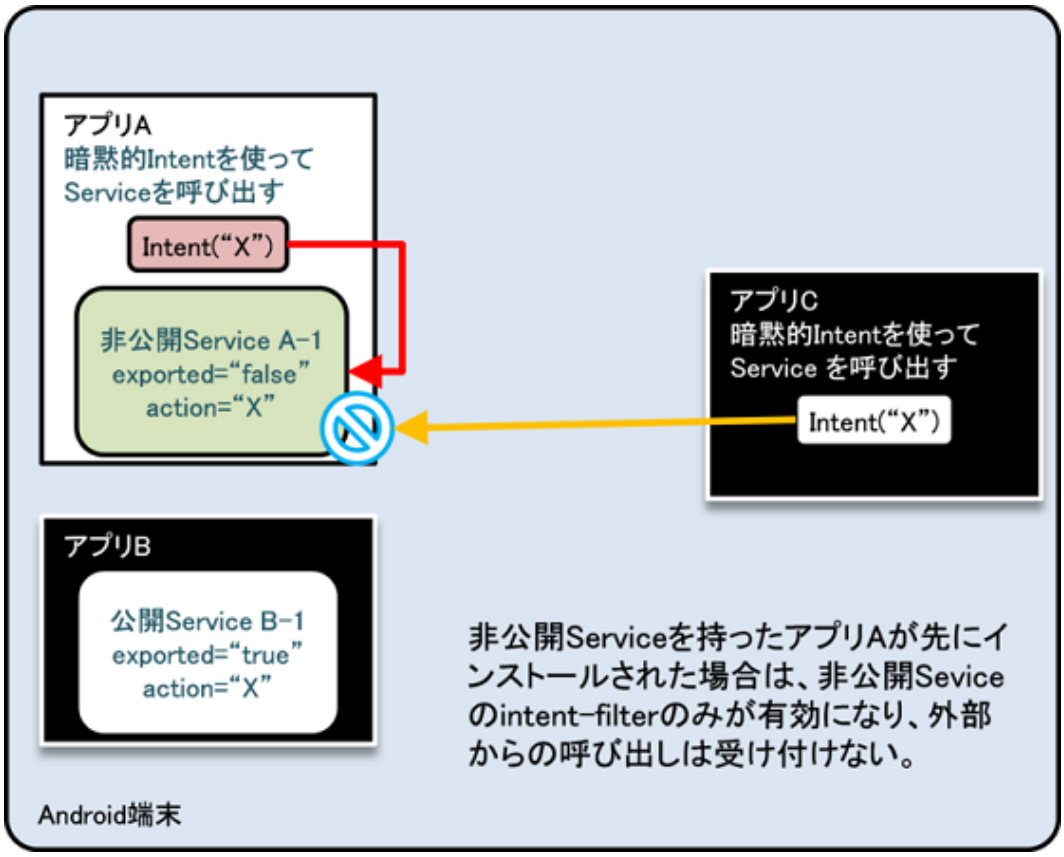

図 4.4.5　アプリ A → アプリ B の順でインストールされた場合

図 4.4.6 は、アプリ B →アプリ A の順でインストールされた場合であり、セキュリティ面からみて問題がある。アプリ A が暗黙的 Intent を送信して同一アプリ内の非公開 Service を呼び出そうとするが、先にインストールしたアプリ B の公開 Activity(B-1) が呼び出されてしまう例を示している。これによりアプリ A からアプリ B に対してセンシティブな情報を送信する可能性が生じてしまう。アプリ B がマルウェアであれば、そのままセンシティブな情報の漏洩に繋がる。

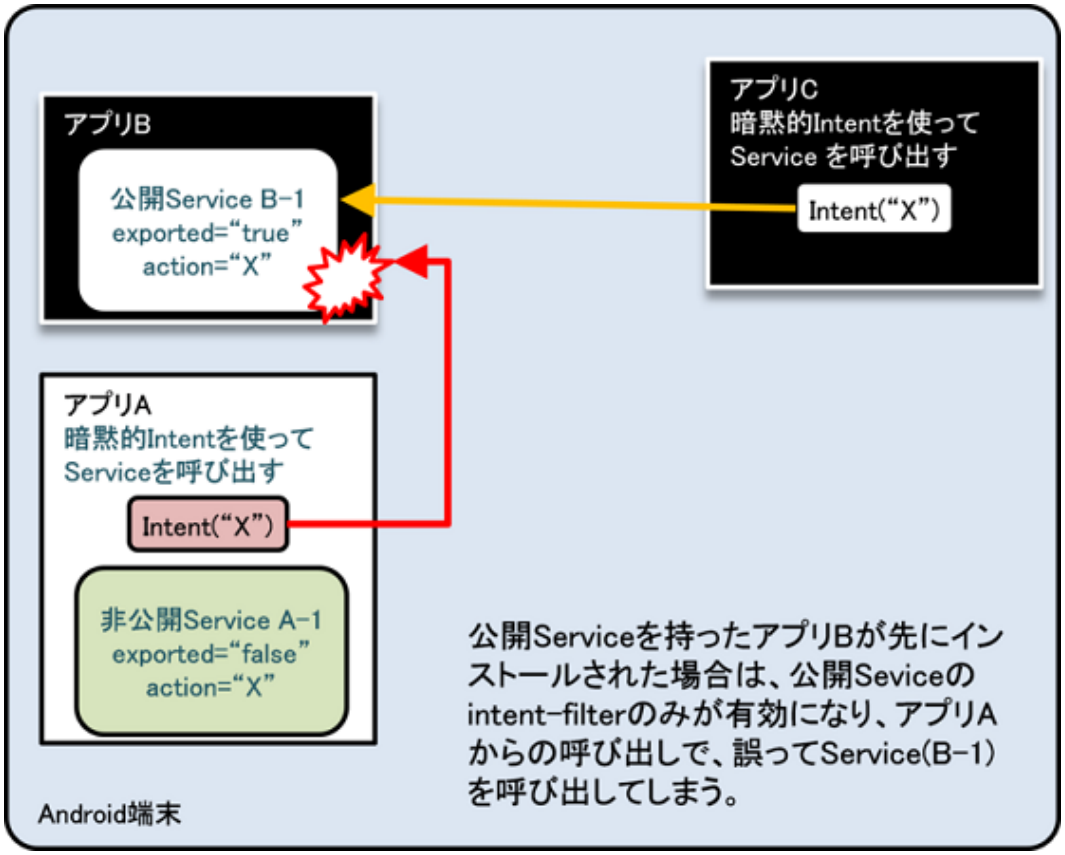

図 4.4.6　アプリ B → アプリ A の順でインストールされた場合

このように、Intent Filter を用いた非公開 Service の暗黙的 Intent 呼び出しは、意図せぬアプリの呼び出しや意図せぬアプリへのセンシティブな情報の送信を避けるためにも行うべきではない。

なお、Android 12 以降、exported 設定は必須となっている。詳細は、4.7.3.1. コンポーネント公開制御と *Intent* 送信の制限」を参照すること。

4.4.3.2 Service の実装方法について

Service の実装方法は多様であり、サンプルコードで分類したセキュリティ上のタイプとの相性もあるため簡単に特徴を示す。startService を利用する場合と bindService を利用する場合とに大きく分かれるが、startService と bindService の両方で利用できる Service を作成することも可能である。Service の実装方法を決定するために、次のような項目について検討を行うことになる。

- Service を別アプリに公開するか（Service の公開）
- 実行中にデータのやり取りを行うか (データの相互送受信)
- Service を制御するか（起動や終了など）
- 別プロセスとして実行するか（プロセス間通信）
- 複数の処理を同時に行うか (並行処理)

実装方法の分類と各々の項目の実現の可否を表にすると、表 4.4.3 のようになる。x は実現不可能かもしくは提供される機能とは別の枠組みが必要な場合を表す。

表 4.4.3　Service の実装方法の分類

分類	**Service** の公開	データの相互送受信	**Service** の制御（起動・終了）	プロセス間通信	並行処理
startService 型	o	x	o	o	x
IntentService 型	o	x	x	o	x
local bind 型	x	o	o	x	x
Messen-gerbind 型	o	o	o	o	x
AIDLbind 型	o	o	o	o	o

startService 型

最も基本的な Service である。Service クラスを継承し、onStartCommand で処理を行う Service のことを指す。

利用する側は、Service を Intent で指定して startService を使用して呼び出す。Intent の送信元に対して、結果などのデータを直接返すことはできないため、Broadcast など別の方法を組み合わせて実現する必要がある。具体的な実装例は、「4.4.1.1. 非公開 *Service* を作る・利用する」を参照のこと。

セキュリティ上のチェックは onStartCommand で行う必要があるが、送信元のパッケージ名が取得できないためパートナー限定 Service には使用できない。

IntentService 型

IntentService は Service を継承して作られているクラスである。呼び出し方は、startService 型と同様である。通常の Service（startService 型）に比べて以下の特徴がある。

- Intent の処理は onHandleIntent で行う。（onStartCommand は使わない）
- 別スレッドで実行される
- 処理がキューイングされる

処理が別スレッドのため呼び出しは即座に返され、キューイング機構によりシーケンシャルに Intent に対する処理が行われる。各 Intent の並行処理はされないが、製品の要件によっては実装の簡素化の一つとして選択が可能である。Intent の送信元に対して、結果などのデータを直接返すことはできないため、Broadcast など別の方法を組み合わせて実現する必要がある。具体的な実装例は、「4.4.1.2. 公開 *Service* を作る・利用する」を参照のこと。

セキュリティ上のチェックは onHandleIntent で行う必要があるが、送信元のパッケージ名が取得できないためパートナー限定 Service には使用できない。

local bind 型

アプリと同じプロセス内でのみ動くローカル Service を実装するための方法を指す。Binder クラスから派生したクラスを定義して、Service で実装した機能（メソッド）を呼び出し元に提供できるようにする。

利用する側は、Service を Intent で指定して bindService を使用して呼び出す。Service を bind する方法の中では、最もシンプルな実装であるが、別プロセスでの起動や Service の公開ができないため用途は限定される。具体的な実装例は、サンプルコードに含まれるプロジェクト「Service PrivateServiceLocalBind」を参照のこと。

セキュリティ的には非公開 Service のみ実装可能である。

Messenger bind 型

Messenger の仕組みを利用して Service との連携を実現する方法を指す。

Service を利用する側からも Message の返信先として Messenger を渡すことができるため、双方でのデータのやり取りが比較的容易に実現可能である。また、処理はキューイングされるため、スレッドセーフに動作する特徴がある。各 Message の並行処理はされないが、製品の要件によっては実装の簡素化の一つとして選択が可能である。利用する側は、Service を Intent で指定して bindService を使用して呼び出す。具体的な実装例は、「4.4.1.4. 自社限定 *Service*」を参照のこと。

セキュリティ上のチェックは onBind や Message Handler で行う必要があるが、送信元のパッケージ名が取得できないためパートナー限定 Service には使用できない。

AIDL bind 型

AIDL の仕組みを利用して Service との連携を実現する方法を指す。AIDL によってインターフェースを定義し、Service の持つ機能をメソッドとして提供する。また、AIDL で定義したインターフェースを利用側で実装することで、コールバックを実現することもできる。マルチスレッド呼び出しは可能だが、排他処理はされないので Service 側で明示的に実装する必要がある。

利用する側は、Service を Intent で指定して bindService を使用して呼び出す。具体的な実装例は、「4.4.1.3. パートナー限定 *Service*」を参照のこと。

セキュリティ上のチェックは自社限定 Service では onBind で、パートナー限定 Service では AIDL で定義したインターフェースの各メソッドで行う必要がある。本文書で分類した全セキュリティタイプの Service に利用可能である。

4.4.3.3 必須となった Service タイプ指定

たとえばあるウィンドウでウェブをブラウジングしているときに、別のアプリで音楽を再生する場合など、同時に実行するアプリが多いほど、システムには大きな負荷がかかる。そのため、Android 8 以降、バックグラウンドアプリからバックグラウンド Service の作成は許可されなくなっており、ユーザーが認識しやすいフォアグラウンド Service として作成するよう、Context.startForegroundService() メソッドが追加になった。

startForegroundService で Service を作成した場合、当該の Service が動作中は通知ドロワーに Service が起動中である旨をユーザーに通知する必要がある。

以下は、フォアグラウンド Service で MediaPlayer を使用し音楽を再生するコードの抜粋である。

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);
    // 中略
    buttonStart.setOnClickListener( v -> {
        Intent intent = new Intent(getApplication(), MediaPlaybackService.class);
        intent.putExtra("REQUEST_CODE", 1);
```

(次のページに続く)

(前のページからの続き)

```
        // The launcher with the Intent you want to start
        requestPermissionLauncher.launch(Manifest.permission.POST_NOTIFICATIONS);

        // Start Service
        startForegroundService(intent);
    });
```

このコードは Android 13 までは問題なく動作するが、Android 14 からは更に、フォアグラウンド Service タイプの明示的な指定が必須となった。

Service タイプの指定は AndroidManifest.xml の<service>要素内で行う。修正例は以下の通り。

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" />



    <!--中略-->

    <service
        android:name=".MediaPlaybackService"
        android:foregroundServiceType="mediaPlayback"
        android:exported="false">
    </service>
```

Android 14 をターゲットとするアプリで Service タイプを指定しなかった場合、startForegroundService 実行後、Service 内で startForeground が実行された際に例外が発生する。

```
java.lang.RuntimeException: Unable to start service com.example.myapplication.
↪MediaPlaybackService@b675484 with Intent { cmp=com.example.myapplication/.
↪MediaPlaybackService (has extras) }: java.lang.SecurityException: Starting FGS with type␣
↪mediaPlayback callerApp=ProcessRecord{cff2a18 14391:com.example.myapplication/u0a193}␣
↪targetSDK=34 requires permissions: all of the permissions allOf=true [android.permission.
↪FOREGROUND_SERVICE_MEDIA_PLAYBACK]
```

以下は、指定が必須となった Service タイプの一覧である。

- camera
- connectedDevice
- dataSync
- health
- location
- mediaPlayback

- mediaProjection
- microphone
- phoneCall
- remoteMessaging
- shortService
- specialUse
- systemExempted

4.4.3.4 フォアグラウンド サービスの変更

Android 15 では、セキュリティとプライバシーの向上を目的として、複数のフォアグラウンドサービスに関する動作変更が導入された。これらの変更は、アプリケーションがシステムリソースを不適切に使用することを防ぎ、ユーザーの意図しない動作を減少させることを目指している。以下に、主要な変更内容とそれに対するセキュリティ対策を示す。

データ同期フォアグラウンドサービスのタイムアウト動作

Android 15 では、dataSync フォアグラウンドサービスに対して新しいタイムアウト動作が導入された。この結果、dataSync サービスは 24 時間以内に合計 6 時間の実行が許可され、それを超えると Service.onTimeout(int, int) メソッドが呼び出される。

この変更は、長時間実行されるフォアグラウンドサービスがシステムリソースを占有し続けることで、他のアプリケーションやシステム全体のパフォーマンスに悪影響を与えることを防ぐためである。また、ユーザーの意図しないデータ操作が長時間続くことで、プライバシーが侵害されるリスクも軽減するためである。

対策

1. タイムアウトメソッドの実装:

```
override fun onTimeout(startId: Int, timeout: Int) {
    stopSelf(startId)
}
```

2. サービスの実行時間制限: アプリの dataSync サービスが 24 時間以内に合計 6 時間を超えて実行されないように設計する。ユーザーがアプリをフォアグラウンドに操作した場合にのみ、タイマーがリセットされることを確認する。
3. 代替 API の利用: dataSync フォアグラウンドサービスを使用せず、WorkManager などの代替 API を利用することで、バックグラウンド処理の効率性とセキュリティを向上させる。

新しいメディア処理フォアグラウンドサービス タイプ

Android 15 では、mediaProcessing フォアグラウンドサービスタイプが導入され、メディアファイルのコード変換などに適している。このサービスも 24 時間以内に合計 6 時間の実行が許可され、それを超えると Service.onTimeout(int, int) メソッドが呼び出される。

この変更は、長時間のメディア処理がシステムリソースを過剰に使用し、他のアプリケーションやシステム全体のパフォーマンスに悪影響を与えることを防ぐためである。

対策

1. タイムアウトメソッドの実装:

```
override fun onTimeout(startId: Int, timeout: Int) {
    stopSelf(startId)
}
```

2. サービスの実行時間制限：アプリの mediaProcessing サービスが 24 時間以内に合計 6 時間を超えて実行されないように設計する。ユーザーがアプリをフォアグラウンドに操作した場合にのみ、タイマーがリセットされることを確認する。
3. 代替 API の利用：mediaProcessing フォアグラウンドサービスを使用せず、WorkManager などの代替 API を利用することで、バックグラウンド処理の効率性を向上させる。
4. メディア処理の分割：長時間のメディア処理を小さなタスクに分割し、各タスクが制限時間内に完了するようにする。
5. ユーザー通知：長時間のメディア処理が必要な場合、ユーザーに進捗状況を通知し、必要に応じて処理を一時停止または再開できるようにする。

BOOT_COMPLETED ブロードキャストレシーバによるフォアグラウンドサービスの起動に関する制限

Android 15 では、BOOT_COMPLETED ブロードキャストレシーバによる特定のフォアグラウンドサービス（例: dataSync, camera, mediaPlayback など）の起動が制限される。

セキュリティリスクこの変更は、システム起動時に自動的に開始されるサービスが、ユーザーの意図しないバックグラウンド動作を引き起こし、プライバシー侵害やリソースの過剰消費につながることを防ぐためである。

対策

1. 起動制限の遵守: BOOT_COMPLETED レシーバーを使用して制限対象のフォアグラウンドサービスを起動しないように設計する。
2. 代替手段の検討: 必要な場合、ユーザーの明確な操作や許可を得た上でサービスを開始するようにする。

SYSTEM_ALERT_WINDOW 権限を保持しているときのフォアグラウンドサービスの起動に関する制限

Android 15 では、アプリが SYSTEM_ALERT_WINDOW 権限を持っている場合、フォアグラウンドサービスを開始する前に TYPE_APPLICATION_OVERLAY ウィンドウを表示する必要がある。

この変更は、SYSTEM_ALERT_WINDOW 権限を持つアプリがバックグラウンドから自由にフォアグラウンドサービスを起動することにより、ユーザーの意図しない操作や悪意のある動作が発生するリスクを防ぐためである。

対策

1. オーバーレイウィンドウの表示確認: アプリがフォアグラウンドサービスを開始する前に、TYPE_APPLICATION_OVERLAY ウィンドウが表示されていることを確認する。

```
// 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
```

(次のページに続く)

（前のページからの続き）

```
    showOverlayWindow()
    // Optionally, you could delay the start of the service until the overlay is visible
}
```

2. 可視性の監視: View.onWindowVisibilityChanged をオーバーライドして、ウィンドウの可視性が変更されるたびに通知を受け取るようにする。

```
override fun onWindowVisibilityChanged(visibility: Int) {
    super.onWindowVisibilityChanged(visibility)
    // Handle visibility changes
}
```

3. 処理順序の確認: アプリの処理順序を確認し、バックグラウンドからフォアグラウンドサービスを開始する前にアクティブなオーバーレイウィンドウが存在することを保証する。

4.4.3.5 バックグラウンドオーディオの制限強化

Android 17 以降では、ユーザーの意図しないバックグラウンド再生を防止するため、オーディオ再生、オーディオフォーカス要求、音量変更 API に対する制限が強化されている。この制限は、アプリの targetSdkVersion に関係なく、Android 17 上で実行されるすべてのアプリに適用される。

アプリがバックグラウンド状態（有効なライフサイクルにない状態）にあるときにこれらの音声 API を呼び出すと、例外がスローされることなくサイレントに失敗（ブロック）する。例外でクラッシュしないため、実装によっては「処理は通過しているのになぜか音が鳴らない」という形で不具合化しやすい点に注意が必要である。また、オーディオフォーカス API の場合は、結果コードとして AUDIOFOCUS_REQUEST_FAILED が返されて失敗する。

バックグラウンドで音声を扱うためには、以下の条件を満たすサービス構成が必要となる。

- Android 17 上で実行されるすべてのアプリ： 画面上に見えている Activity が存在するか、 SHORT_SERVICE 以外のフォアグラウンドサービス（FGS）が実行されている必要がある。
- Android 17 以上をターゲットとするアプリ： 実行するフォアグラウンドサービスが while-in-use （使用中のみ許可）の要件を満たしている必要がある。つまり、ユーザーの操作に応じて開始されたか、アプリが画面に表示されている間に FGS が開始されていなければならない。

ただし、アプリが正確なアラームの権限（ SCHEDULE_EXACT_ALARM など）を保持しており、かつ USAGE_ALARM のオーディオストリームを使用して再生する場合に限り、例外的に while-in-use 要件は免除される。

通話系のアプリや、画面表示中のみ再生を行うアプリへの影響は比較的小さいが、音楽やポッドキャストなどをバックグラウンドで長時間再生するアプリでは、ユーザーの明示的な操作をトリガーとして、適切なメディア再生用のフォアグラウンドサービスを開始する設計を徹底する必要がある。端末起動直後にいきなり FGS を立ち上げて音を鳴らすようなフローは抑止対象となりやすいため、避けるべきである。

以下は、Android 17 の制限によってバックグラウンドでの音声再生がどのようにブロックされるかを検証するためのコード例である。

ボタンタップなどの操作をトリガーとしてこの関数を呼び出した直後にホーム画面に戻り、アプリを完全にバックグラウンドへ移行させる。指定した時間（ここでは 10 秒）が経過しても音声は再生されず、SecurityException などの例外も発生しない。Logcat 上では mediaPlayer went away with unhandled events のようなシステムログのみが出力され、サイレントに失敗することが確認できる。

```
// 指定した時間（ミリ秒）後に音声を再生する処理
fun playAudioWithDelay(context: Context, delayMillis: Long) {
    Handler(Looper.getMainLooper()).postDelayed({
        try {
            // システムのデフォルト通知音を取得（音源として利用するだけ）
            val defaultUri = RingtoneManager.getDefaultUri(RingtoneManager.TYPE_
↪NOTIFICATION)

            // AudioAttributes を明示的に「メディア（音楽）再生」として設定し、
            // アラーム用途による制限免除を回避して純粋なメディア再生としてテストする
            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() }
            }

            // 再生開始（Android 17 以降でバックグラウンド状態ならここでサイレントに失敗する）
            mediaPlayer.start()
            Log.d("AudioTest", "MediaPlayer.start() を実行しました")
        } catch (e: Exception) {
            Log.e("AudioTest", "再生エラー: ${e.message}")
        }
    }, delayMillis)
}
```

4.5 SQLite を使う

本文書では SQLite を使用してデータベースの作成および操作を行う際にセキュリティ上で注意すべき点をまとめる。主なポイントは、データベースファイルのアクセス権の適切な設定と SQL インジェクションに対する対策である。ここでは、直接外部からデータベースファイルの読み書きを許す (複数アプリで共有する) ようなデータベースはここでは想定せず Content Provider のバックエンドやアプリ単体での使用を前提とする。また、ある程度センシティブな情報を扱っていることを想定しているが、そうでない場合も他アプリからの想定外の読み書きを避けるためにもここで挙げる対策を適用することをお勧めする。

4.5.1 サンプルコード

4.5.1.1 データベースの作成と操作

Android のアプリでデータベースを扱う場合、SQLiteOpenHelper を使用することでデータベースファイルの適切な配置およびアクセス権の設定（他のアプリがアクセスできない設定）ができる[*16]。ここでは、アプリ起動時にデータベースを作成し、UI 上からデータの検索・追加・変更・削除を行う簡単なアプリを例に、外部からの入力に対して不正な SQL が実行されないように SQL インジェクション対策したサンプルコードを示す。

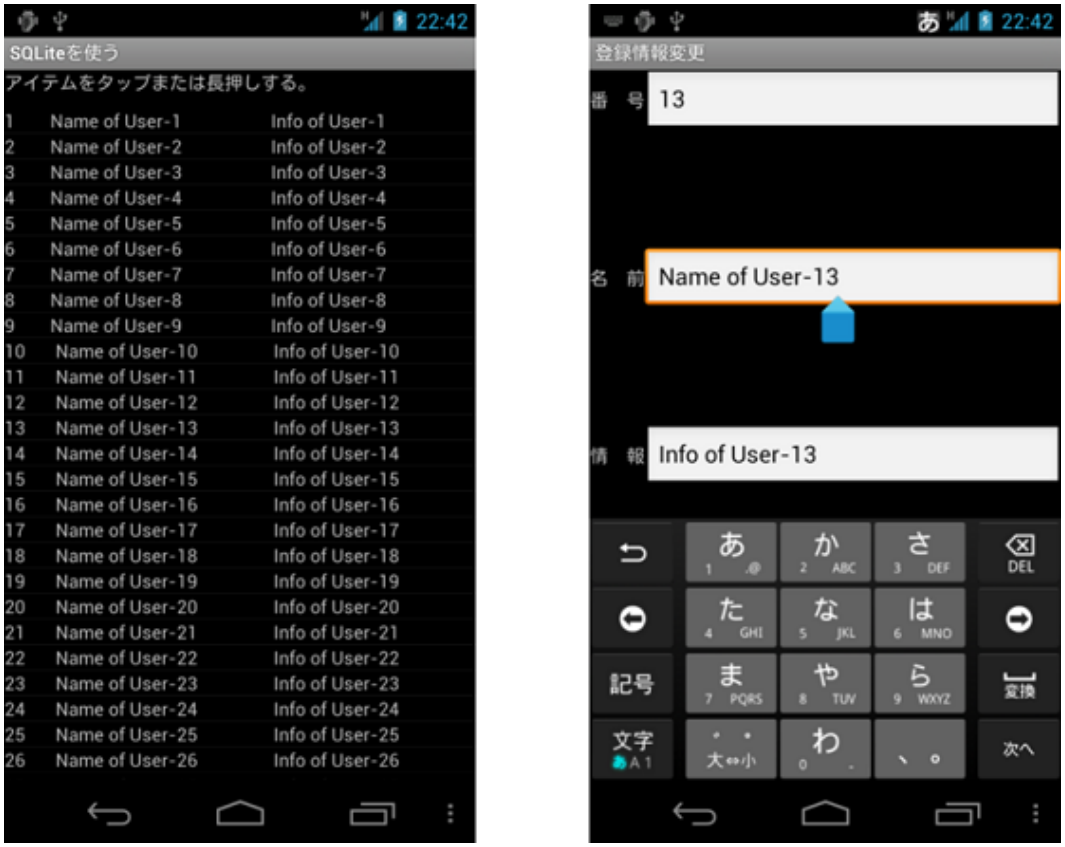

図 4.5.1　Android のアプリでデータベースを扱う

ポイント：

1. データベース作成には SQLiteOpenHelper を使用する
2. SQL インジェクションの対策として入力値を SQL 文に使用する場合にはプレースホルダを利用する
3. SQL インジェクションの保険的な対策としてアプリ要件に従って入力値をチェックする

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.
```

(次のページに続く)

[*16] ファイルの配置に関しては、SQLiteOpenHelper のコンストラクタの第 2 引数（name）にファイルの絶対パスも指定できる。そのため、誤って SD カードを直接指定した場合には他のアプリからの読み書きが可能になるので注意が必要である。

(前のページからの続き)

```
 * 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;
import android.util.Log;
import android.widget.Toast;

public class SampleDbOpenHelper extends SQLiteOpenHelper {
    private SQLiteDatabase  mSampleDb;  //取り扱うデータを格納するデータベース

    public static SampleDbOpenHelper newHelper(Context context)
    {
        //★ポイント 1 ★ データベース作成には SQLiteOpenHelper を使用する
        return new SampleDbOpenHelper(context);
    }

    public SQLiteDatabase getDb() {
        return mSampleDb;
    }

    //Writable モードで DB を開く
    public void openDatabaseWithHelper() {
        try {
            if (mSampleDb != null && mSampleDb.isOpen()) {
                if (!mSampleDb.isReadOnly())// 既に読み書き可能でオープン済み
                    return;
                mSampleDb.close();
            }
            mSampleDb  = getWritableDatabase(); //この段階でオープンされる
        } catch (SQLException e) {
            //データベース構築に失敗した場合ログ出力
            Log.e(mContext.getClass().toString(),
```

(次のページに続く)

(前のページからの続き)

```
                mContext.getString(R.string.DATABASE_OPEN_ERROR_MESSAGE));
            Toast.makeText(mContext,
                           R.string.DATABASE_OPEN_ERROR_MESSAGE,
                           Toast.LENGTH_LONG).show();
        }
    }

    //ReadOnly モードで DB を開く
    public void openDatabaseReadOnly() {
        try {
            if (mSampleDb != null && mSampleDb.isOpen()) {
                if (mSampleDb.isReadOnly())// 既に ReadOnly でオープン済み
                    return;
                mSampleDb.close();
            }
            SQLiteDatabase.openDatabase(mContext.getDatabasePath(CommonData.DBFILE_NAME).
↪getPath(),
                                        null, SQLiteDatabase.OPEN_READONLY);
        } catch (SQLException e) {
            //データベース構築に失敗した場合ログ出力
            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
    public void closeDatabase() {
        try {
            if (mSampleDb != null && mSampleDb.isOpen()) {
                mSampleDb.close();
            }
        } catch (SQLException e) {
            //データベース構築に失敗した場合ログ出力
            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();
        }
    }

    //Context を覚えておく
    private Context mContext;
```

(次のページに続く)

(前のページからの続き)

```
    //テーブル作成コマンド
    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);  //DB構築コマンドの実行
        } catch (SQLException e) {
            //データベース構築に失敗した場合ログ出力
            Log.e(this.getClass().toString(),
                  mContext.getString(R.string.DATABASE_CREATE_ERROR_MESSAGE));
        }
    }

    @Override
    public void onUpgrade(SQLiteDatabase arg0, int arg1, int arg2) {
        // データベースのバージョンアップ時に実行される、データ移行などの処理を記述する
    }
}
```

DataSearchTask.java(SQLite Database プロジェクト)

```
/*
 * 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,
```

(次のページに続く)

(前のページからの続き)

```
 * 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;


//データ検索タスク
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;

        //★ポイント 3★ SQL インジェクションの保険的な対策としてアプリ要件に従って入力値をチェックする
        if (!DataValidator.validateData(idno, name, info))
            {
                return null;
            }

        //引数が全部 null だったら全件検索する)
```

(次のページに続く)

(前のページからの続き)

```
        if ((idno == null || idno.length() == 0) &&
            (name == null || name.length() == 0) &&
            (info == null || info.length() == 0) ) {
            try {
                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;
        }

        //No が指定されていたら No で検索
        if (idno != null && idno.length() > 0) {
            String selectionArgs[] = {idno};

            try {
                //★ポイント 2★ SQL インジェクションの対策として入力値を SQL 文に使用する場合にはプレースホルダを使用する
                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;
        }

        //Name が指定されていたら Name で完全一致検索
        if (name != null && name.length() > 0) {
            String selectionArgs[] = {name};
            try {
                //★ポイント 2★ SQL インジェクションの対策として入力値を SQL 文に使用する場合にはプレースホルダを使用する
                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;
```

(次のページに続く)

(前のページからの続き)

```
        }

        //それ以外の場合は info を条件にして部分一致検索
        //入力として受け取った info 内の$をエスケープ
        String argString = info.replaceAll("@", "@@");
        //入力として受け取った info 内の % をエスケープ
        argString = argString.replaceAll("%", "@%");
        //入力として受け取った info 内の_をエスケープ
        argString = argString.replaceAll("_", "@_");
        String selectionArgs[] = {argString};

        try {
            //★ポイント 2 ★ SQL インジェクションの対策として入力値を SQL 文に使用する場合にはプレースホルダを使用する
            cur = mSampleDB.query(CommonData.TABLE_NAME, cols,
                        "info LIKE '%' || ? || '%' ESCAPE '@'",
                                selectionArgs, null, null, null);
        } 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.
```

(次のページに続く)

(前のページからの続き)

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

package org.jssec.android.sqlite;

public class DataValidator {
    //入力値をチェックする
    //数字チェック
    public static boolean validateNo(String idno) {
        //null、空文字は OK
        if (idno == null || idno.length() == 0) {
            return true;
        }

        //数字であることを確認する
        try {
            if (!idno.matches("[1-9][0-9]*")) {
                //数字以外の時はエラー
                return false;
            }
        } catch (NullPointerException e) {
            //エラーを検出した
            return false;
        }

        return true;
    }

    // 文字列の長さを調べる
    public static boolean validateLength(String str, int max_length) {
        //null、空文字は OK
        if (str == null || str.length() == 0) {
            return true;
        }

        //文字列の長さが MAX 以下であることを調べる
        try {
            if (str.length() > max_length) {
                //MAX より長い時はエラー
                return false;
            }
        } catch (NullPointerException e) {
            //バグ
            return false;
```

(次のページに続く)

(前のページからの続き)

```
        }

        return true;
    }

    // 入力値チェック
    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 ルールブック

SQLite を使用する際には以下のルールを守ること。

1. *DB* ファイルの配置場所、アクセス権を正しく設定する（必須）
2. 他アプリと *DB* データを共有する場合は *Content Provider* でアクセス制御する（必須）
3. *DB* 操作時に可変パラメータを扱う場合はプレースホルダを使用する（必須）)

4.5.2.1 DB ファイルの配置場所、アクセス権を正しく設定する（必須）

DB ファイルのデータの保護を考えた場合、DB ファイルの配置場所とアクセス権の設定は合わせて考慮すべき重要な要素である。

例えば、ファイルのアクセス権を正しく設定したつもりでも、SD カードなどアクセス権の設定を行えない場所に配置している場合には、誰からでもアクセス可能な DB ファイルになってしまう。また、アプリディレクトリに配置した場合でも、アクセス権を正しく設定しないと意図しないアクセスを許してしまうことになる。ここでは、配置場所とアクセス権設定について守るべき点を挙げた後、それを実現するための方法について説明する。

まず配置場所とアクセス権設定については、DB ファイル (データ) を保護する観点から考えると、以下の 2 点を実施する必要がある。

1. 配置場所

Context#getDatabasePath(String name) で取得できるファイルパスや場合によっては Context#getFilesDir で取得できるディレクトリの場所に配置する[*17]

[*17] どちらのメソッドも該当するアプリだけが読み書き権限を与えられ、他のアプリからはアクセスができないディレクトリ（パッケージディレク

2. アクセス権

MODE_PRIVATE（=ファイルを作成したアプリのみがアクセス可能）モードに設定する

この 2 点を実施することで、他のアプリからアクセスできない DB ファイルの作成を行うことができる。これらを実施するためには以下の方法が挙げられる。

1. SQLiteOpenHelper を使用する

2. Context#openOrCreateDatabase を使用する

DB ファイルの作成に際しては、SQLiteDatabase#openOrCreateDatabase を使用することもできる。しかし、このメソッドを使用した場合、Android スマートフォンの機種によっては、他のアプリから読み取り可能な DB ファイルが作成されることが分かっている。そのため、このメソッドの使用は避けて、他の方法を利用することを推奨する。上に挙げた 2 つの方法について、それぞれの特徴を以下で説明する。

SQLiteOpenHelper を使用する

SQLiteOpenHelper を使用する場合、開発者はあまり多くのことを考えなくてもよい。SQLiteOpenHelper を派生したクラスを作成し、コンストラクタの引数に DB の名前（ファイル名に使われる）[*18] を指定すれば、自動的に上記のセキュリティ要件を満たす DB ファイルを作成してくれる。

「4.5.1.1. データベースの作成と操作」に具体的な使用方法を示しているので参照すること。

Context#openOrCreateDatabase を使用する

Context#openOrCreateDatabase メソッドを使用して DB の作成を行う場合、ファイルのアクセス権をオプションで指定する必要があり、明示的に MODE_PRIVATE を指定する。

ファイルの配置に関しては、DB 名（ファイル名に使用される）の指定を SQLiteOpenHelper と同様に行えるので、自動的に前述のセキュリティ要件を満たすファイルパスにファイルが作成される。ただし、フルパスも指定できるので SD カードなどを指定した場合、MODE_PRIVATE を指定しても他アプリからアクセス可能になってしまうため注意が必要である。

DB に対して明示的にアクセス許可設定を行う例：MainActivity.java

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

    //データベースの構築
    try {
        //MODE_PRIVATE を設定して DB を作成
        db = Context.openOrCreateDatabase("Sample.db", MODE_PRIVATE, null);
    } catch (SQLException e) {
        //データベース構築に失敗した場合ログ出力
        Log.e(this.getClass().toString(), getString(R.string.DATABASE_OPEN_ERROR_MESSAGE));
        return;
    }
```

(次のページに続く)

トリ）のサブディレクトリ以下のパスが取得できる。

[*18] （ドキュメントに記述はないが）SQLiteOpenHelper の実装では DB の名前にはファイルのフルパスを指定できるので、SD カードなどアクセス権の設定できない場所のパスが意図せず入力されないように注意が必要である。

(前のページからの続き)

```
    //省略 その他の初期化処理
}
```

なお、アクセス権の設定は MODE_PRIVATE と合わせて以下の 3 種類があり、MODE_WORLD_READABLE と MODE_WORLD_WRITEABLE は OR 演算で同時指定することもできる。ただし、MODE_PRIVATE 以外は API Level 17 以降では deprecated となっており、API Level 24 以降ではセキュリティ例外が発生する。API Level 15 以降を対象とする場合でも、通常はこのフラグを使用しないことが望ましい[*19]。

- MODE_PRIVATE 作成アプリのみ読み書き可能
- MODE_WORLD_READABLE 作成アプリは読み書き可能、他は読み込みのみ
- MODE_WORLD_WRITEABLE 作成アプリは読み書き可能、他は書き込みのみ

4.5.2.2 他アプリと DB データを共有する場合は Content Provider でアクセス制御する（必須）

他のアプリと DB データを共有する手段として、DB ファイルを WORLD_READABLE、WORLD_WRITABLE として作成し、他のアプリから直接アクセスできるようにするという方法がある。しかし、この方法では DB にアクセスするアプリや DB への操作を制限できないため、意図しない相手（アプリ）にデータを読み書きされることもある。結果として、データの機密性や整合性に問題が生じたり、マルウェアの攻撃対象となったりする可能性も考えられる。

以上のことから、Android において DB データを他のアプリと共有する場合は、Content Provider を使うことを強くお勧めする。Content Provider を使うことにより、DB に対するアクセス制御を実現できるというセキュリティの観点からのメリットだけでなく、DB スキーマ構造を Content Provider 内に隠ぺいできるといった設計観点のメリットもある。

4.5.2.3 DB 操作時に可変パラメータを扱う場合はプレースホルダを使用する（必須）

SQL インジェクションを防ぐという意味で、任意の入力値を SQL 文に組み込む時はプレースホルダを使用するべきである。プレースホルダを使用した SQL の実行方法としては以下の 2 つの方法を挙げることができる。

1. SQLiteDatabase#compileStatement() を使用して SQLiteStatement を取得する。その後、SQLiteStatement#bindString()、bindLong() などを使用してパラメータをプレースホルダに配置する
2. SQLiteDatabese クラスの execSQL()、insert()、update()、delete()、query()、rawQuery()、replace() などを呼び出す際にプレースホルダを持った SQL 文を使用する

なお、SQLiteDatabase#compileStatement() を使用して、SELECT コマンドを実行する場合、

「SELECT コマンドの結果として先頭の 1 要素 (1 行 1 列目) しか取得できない」

という制限があるので用途が限られる。

どちらの方式を使う場合でも、プレースホルダに与えるデータの内容は事前にアプリ要件に従ってチェックされていることが望ましい。以下で、それぞれの方法について説明する。

SQLiteDatabase#compileStatement() を使用する場合：

以下の手順でプレースホルダへデータを渡す。

[*19] MODE_WORLD_READABLE および MODE_WORLD_WRITEABLE の性質と注意点については、「4.6.3.2. ディレクトリのアクセス権設定」を参照

1. SQLiteDatabase#compileStatement() を使用してプレースホルダを含んだ SQL 文を SQLiteStatement として取得する。
2. 作成した SQLiteStatement オブジェクトに対して、bindLong()、bindString() などのメソッドを使用してプレースホルダに設定する。
3. SQLiteStatement オブジェクトの execute() などのメソッドによって SQL を実行する。

プレースホルダ使用例：DataInsertTask.java（抜粋）

```
//データ追加タスク
public class DataInsertTask extends AsyncTask<String, Void, Void> {
    private MainActivity    mActivity;
    private SQLiteDatabase  mSampleDB;

    public DataInsertTask(SQLiteDatabase db, MainActivity activity) {
        mSampleDB = db;
        mActivity = activity;
    }

    @Override
    protected Void doInBackground(String... params) {
        String  idno = params[0];
        String  name = params[1];
        String  info = params[2];

        //★ポイント 3★ アプリケーション要件に従って入力値をチェックする
        if (!DataValidator.validateData(idno, name, info))
        {
            return null;
        }
        //★ポイント 2 データ追加処理
        //プレースホルダを使用する
        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();
        }
```

(次のページに続く)

（前のページからの続き）

```
        return null;
    }
    // ～省略～
}
```

あらかじめ実行する SQL 文をオブジェクトとして作成しておきパラメータを当てはめる形である。実行する処理が確定しているので、SQL インジェクションが発生する余地はない。また、SQLiteStatement オブジェクトを再利用することで処理効率を高めることができるというメリットもある。

SQLiteDatabase が提供する各処理用のメソッドを使用する場合：

SQLiteDatabase が提供する DB 操作メソッドには、SQL 文を使用するものとそうでないものがある。SQL 文を使用するメソッドに SQLiteDatabase# execSQL()/rawQuery() などがあり、以下の手順で実行する。

1. プレースホルダを含んだ SQL 文を用意する。
2. プレースホルダに割り当てるデータを作成する。
3. SQL 文とデータを引数として渡して処理用メソッドを実行する。

一方、SQL 文を使用しないメソッドには、SQLiteDatabase#insert()/update()/delete()/query()/replace() などがある。これらを使用する場合には、以下の手順でデータを渡す。

1. DB に対して挿入/更新するデータがある場合には、ContentValues に登録する。
2. ContentValues を引数として渡して、各処理用メソッド（以下の例では SQLiteDatabase#insert()）を実行する。

各処理用メソッド（SQLiteDatabase#insert()）を使用する例

```
private SQLiteDatabase  mSampleDB;
private void addUserData(String idno, String name, String info) {
   //値の妥当性（型、範囲）チェック、エスケープ処理
   if (!validateInsertData(idno, name, info)) {
       //バリデーションを通過しなかった場合、ログ出力
       Log.e(this.getClass().toString(), getString(R.string.VALIDATION_ERROR_MESSAGE));
       return;
   }
    //挿入するデータの準備
    ContentValues insertValues = new ContentValues();
    insertValues.put("idno", idno);
    insertValues.put("name", name);
    insertValues.put("info", info);
    //Insert 実行
    try {
        mSampleDb.insert("SampleTable", null, insertValues);
    } catch (SQLException e) {
        Log.e(this.getClass().toString(), getString(R.string.DB_INSERT_ERROR_MESSAGE));
        return;
    }
}
```

この例では、SQL コマンドを直接記述せず、SQLiteDatabase が提供する挿入用のメソッドを使用している。SQL コマンドを直接使用しないため、この方法も SQL インジェクションの余地はないと言える。

4.5.3 アドバンスト

4.5.3.1 SQL 文の LIKE 述語でワイルドカードを使用する際にエスケープ処理を施す

LIKE 述語のワイルドカード（%、_）を含む文字列をプレースホルダの入力値として使用した場合、そのままだとワイルドカードとして機能するため、必要に応じて事前にエスケープ処理を施す必要がある。必要なケースとしてはワイルドカードを単体の文字（"%" や"_"）として扱いたい場合が当てはまる。

実際のエスケープ処理は、以下のサンプルコードのように ESCAPE 句を使用して行うことができる。

LIKE を利用した場合のエスケープ処理の例

```
//データ検索タスク
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
    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;

        // ～省略～

        //info を条件にして like 検索（部分一致）
        //ポイント：ワイルドカードに相当する文字はエスケープ処理する
        String argString = info.replaceAll("@", "@@"); //入力として受け取った info 内の@をエスケープ
        argString = argString.replaceAll("%", "@%"); //入力として受け取った info 内の % をエスケープ
        argString = argString.replaceAll("_", "@_"); //入力として受け取った info 内の_をエスケープ
        String selectionArgs[] = {argString};

        try {
```

(次のページに続く)

(前のページからの続き)

```
            //ポイント：プレースホルダを使用する
            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 プレースホルダを使用できない SQL コマンドに対して外部入力を使う

テーブルの作成や削除などの DB オブジェクトを処理対象とした SQL 文を実行する場合、テーブル名などの値に対してプレースホルダを使うことはできない。基本的には、プレースホルダの使用できない値に対して、外部から入力された任意の文字列を使用するようなデータベースの設計はすべきでない。

仕様や機能上の制限でプレースホルダを使用できない場合は、入力値に危険が無いかどうか実行前に確認し、必要な処理を施すことが必須となる。

基本的には、

1. 文字列パラメータとして使用する場合、文字のエスケープやクォート処理を施す
2. 数値パラメータとして使用する場合、数字以外の文字が混入していないことを確認する
3. 識別子、コマンドとして使用する場合、1. に加え、使用できない文字が含まれていないことを確認する

を実施する。

> 参照：[「安全な SQL の呼び出し方 (安全なウェブサイトの作り方別冊)」](https://www.ipa.go.jp/security/vuln/documents/website_security_sql.pdf)

4.5.3.3 不用意にデータベースの書き換えが行われないための対策を行う

SQLiteOpenHelper#getReadableDatabase、getWritableDatabase を使用して DB のインスタンスを取得した場合、どちらのメソッドを利用しても DB は読み書き可能な状態でオープンされる[*20]。また、Context#openOrCreateDatabase、

[*20] getReableDatabase は基本的には getWritableDatabase で取得するのと同じオブジェクトを返す。ディスクフルなどの状況で書き込み可能オブジェクトを生成できない場合にリードオンリーのオブジェクトを返すという仕様である（getWritableDatabase はディスクフルなどの状況では実行エラーとなる）。

SQLiteDatabase#openOrCreateDatabase なども同様である。

これは、アプリ操作や実装の不具合により意図せず DB の中身を書き換えてしまう（書き換えられてしまう）可能性を意味している。基本的にはアプリの仕様と実装の範囲で対応できると考えられるが、アプリの検索機能など、読み取りしか必要のない機能を実装する場合は、データベースを読み取り専用でオープンすることで、設計や検証の簡素化ひいてはアプリ品質の向上に繋がる場合があるので、状況に応じて検討をお勧めする。

具体的には、SQLiteDatabase#openDatabase に OPEN_READONLY を指定してデータベースをオープンする。

読み取り専用でデータベースをオープンする

```
// ～省略～
// データベースのオープン（データベースは作成済みとする）
SQLiteDatabase db
        = SQLiteDatabase.openDatabase(SQLiteDatabase.getDatabasePath("Sample.db"),
                                      null, OPEN_READONLY);
```

> 参照：https://developer.android.com/reference/android/database/sqlite/SQLiteOpenHelper.html#getReadableDatabase()

4.5.3.4 アプリの要件に従って DB の入出力データの妥当性をチェックする

SQLite は型に寛容なデータベースであり、DB 上で Integer として宣言されているカラムに対して文字型のデータを格納することが可能である。DB 内のデータは、数値型を含む全てのデータが平文の文字データとして DB 内に格納されている。このため、Integer 型のカラムに対して文字列型の検索（LIKE '%123%' など）を行うことも可能である。また、VARCHAR(100) のようにデータの最大長を記述してもそれ以上の長さのデータが入力可能であるなど、SQLite での値の制限（正当性確認）は期待できない。

このため、SQLite を使用するアプリは、このような DB の特性に注意して予期せぬデータを DB に格納したり取得したりしないようにアプリの要件に従って対処する必要がある。対処の方法としては次の 2 つがある。

1. データをデータベースに格納する際、型や長さなどの条件が一致しているか確認する
2. データベースから値を取得した際、データが想定外の型や長さでないか確認する

以下では、例として入力値が 1 以上の数字であることを検証するコードを示す。

例：入力データが 1 以上の数字であることを確認する（MainActivity.java より抜粋）

```
public class MainActivity extends Activity {

    // ～省略～

    //追加処理
    private void addUserData(String idno, String name, String info) {
        //No のチェック
        if (!validateNo(idno, CommonData.REQUEST_NEW)) {
            return;
        }

        //データ追加処理
```

(次のページに続く)

（前のページからの続き）

```
        DataInsertTask task = new DataInsertTask(mSampleDb, this);
        task.execute(idno, name, info);
    }

    // ～省略～

    private boolean validateNo(String idno, int request) {
        if (idno == null || idno.length() == 0) {
            if (request == CommonData.REQUEST_SEARCH) {
                //検索処理の時は未指定を OK にする
                return true;
            } else {
                //検索処理以外の時は null、空文字はエラー
                Toast.makeText(this,
                  R.string.IDNO_EMPTY_MESSAGE, Toast.LENGTH_LONG).show();
                return false;
            }
        }

        //数字であることを確認する
        try {
            // 1 以上の値
            if (!idno.matches("[1-9][0-9]*")) {
                //数字以外の時はエラー
                Toast.makeText(this,
                  R.string.IDNO_NOT_NUMERIC_MESSAGE, Toast.LENGTH_LONG).show();
                return false;
            }
        } catch (NullPointerException e) {
            //今回のケースではあり得ない
            return false;
        }

        return true;
    }

    // ～省略～
}
```

4.5.3.5 DB に格納するデータについての考察

SQLite では、データをファイルに格納する際に以下のような実装になっている。

- 数値型を含む全てのデータが平文の文字データとして DB ファイル内に格納される
- DB に対してデータの削除を行ってもデータ自体は DB ファイルから削除されない（削除マークが付くのみ）

- データを更新した場合も DB ファイル内には更新前のデータも削除されず残っている

よって、削除された「はず」の情報が DB ファイル内に残ったままの状態になっている可能性がある。この場合でも、本文書に従って対策を施し、Android のセキュリティ機能が有効であれば、他アプリを含む第三者からデータ・ファイルに直接アクセスされる心配はない。ただし、root 権限を奪取されるなど Android の保護機構を迂回してファイルを抜き出される可能性を考えると、ビジネスに大きな影響を与えるデータが格納されている場合には、Android 保護機構に頼らないデータ保護も検討しなければならない。

これらの理由により、端末の root 権限が奪取された場合でも守る必要があるような重要なデータは SQLite の DB にそのまま格納すべきではない。どうしても重要なデータを格納せざるを得ない場合には暗号化したデータを格納する、DB 全体を暗号化する、などの対策が必要となる。

実際に暗号化が必要な場合、暗号化に使う鍵の扱いやコードの難読化など本文書の範囲を超える課題が多いので、現時点でビジネスインパクトの大きなデータを扱うアプリの開発には専門家への相談をお勧めする。

参考として「4.5.3.6. *[参考]SQLite データベースを暗号化する (SQLCipher for Android)*」に、データベースを暗号化するライブラリを紹介しておく。

4.5.3.6 [参考]SQLite データベースを暗号化する (SQLCipher for Android)

SQLCipher は Zetetic LLC が開発した、SQLite データベースの透過的な 256 ビット AES による暗号化を提供するものである。C 言語で実装された SQLite 拡張ライブラリであり、暗号化には OpenSSL を使用している。また Obj-C、Java、Python 等の言語用の API も提供されている。商用バージョンの他にオープンソース版 (Community version と呼ばれている) が存在し、BSD ライセンスで商用利用可能である。Windows や Linux、macOS など様々なプラットフォームに対応しており、モバイルの世界では Android の他、ノキア/QT、アップルの iOS などで広く使用されている。

このうち Android 用にパッケージングされたのが SQLCipher for Android である[*21]。公開されているソースコードからコンパイルして作成することもできるが、aar フォーマットのライブラリ (android-database-sqlcipher-xxxx.aar) としても配布されており、単に利用するのであればこちらの方が便利である[*22]。標準の SQLite の API の一部を SQLCipher に合わせて変更することで、開発者は通常と同じコーディングで暗号化されたデータベースを利用できるようになっている。ここでは aar フォーマットのライブラリの利用方法について簡単に紹介する。

参照：https://www.zetetic.net/sqlcipher/

使い方

Android Studio では以下の手順で SQLCipher の利用が可能になる。

1. アプリの libs ディレクトリに android-database-sqlcipher-3.5.9.aar を配置する https://www.zetetic.net/sqlcipher/open-source/
2. app/gradle に依存関係を記載

```
dependencies {
    :
    implementation 'net.zetetic:android-database-sqlcipher:3.5.9@aar'
    :
}
```

[*21] https://github.com/sqlcipher/android-database-sqlcipher

[*22] ここで xxxx はライブラリのバージョン番号で、本記事執筆時点の最新バージョンは 3.5.9 である。以下ではこのバージョンを想定して記載している。

3. 通常の android.database.sqlite.* のかわりに、net.sqlcipher.database.* をインポートする (android.database.Cursor はそのまま変更なしに利用できる)

4. データベースを利用する前にライブラリをロード・初期化し、データベースをオープンする際にパスワードを指定する

下に掲げたコードは、データベースを利用するための初期化処理を行うものである。ある Activity がデータベースを利用する前に SQLCipherInitializer.Initialize() を呼ぶ事を想定している。最初に SQLiteDatabase.loadLibs(this) を呼び出し、必要なライブラリをロードし初期化している。また、SQLiteDatabase.openOrCreateDatabase() でデータベースをオープンする際に、パスワード を渡している。データベースはここで与えたパスワードをベースとして生成された暗号鍵を用いて暗号化される。この場合注意すべきなのは平テキストで作成されたデータベースを、後で暗号化したものにすることはできない点であり、データベースを作成する時点でパスワードを指定しなければならない。

```
package android.jssec.org.samplesqlcipher;
import android.content.Context;
// 通常の android.database.sqlite.* のかわりに、net.sqlcipher.database.* をインポートする
import net.sqlcipher.database.SQLiteDatabase;
import java.io.File;
public class SQLCipherInitializer {
    static SQLiteDatabase Initialize(Context ctx, String dbName, String password) {
        // DB を利用する前に必要なライブラリをロードし初期化する
        SQLiteDatabase.loadLibs(ctx);
        // パッケージにローカルの DB 用ディレクトリにデータベースファイルを作成する
        File databaseFile = ctx.getDatabasePath(dbName);
        // DB をオープンする際に暗号化用のパスワードを指定する
        return SQLiteDatabase.openOrCreateDatabase(databaseFile, password, null);
    }
}
```

上は SQLiteDatabase.openOrCreateDatabase() を用いた例であるが、SQLiteOpenHelper#getWritableDatabase() や SQLiteOpenHelper#getReadableDatabase() API が変更になっており、引数としてパスワードを渡すことができるようになっている。いずれの場合もパスワードに null を指定した場合はデータベースは暗号化されず通常の SQLite データベースが作成される。

そのほか注意すべき点として、Context#openOrCreateDatabase() が使えないことである。そのためデータベースファイルの保護モードを設定したりパッケージのローカルディレクトリにデータベースを作成することを強制することはできない。従って SQLiteDatabase.openOrCreateDatabase() でデータベースを作成する場合は、最低限上の例のように getDatabasePath() により自身のパッケージのデータベース用ディレクトリにデータベースを作成することが推奨される。一方 SQLiteOpenHelper のコンストラクタの API には変更はなく、これを用いると android.database.sqlite.SQLiteOpenHelper と同様にパッケージのローカルディレクトリ内にデータベースが作成される。

4.6 ファイルを扱う

Android のセキュリティ設計思想に従うと、ファイルは情報を永続化又は一時保存 (キャッシュ) する目的にのみ利用し、原則非公開にするべきである。アプリ間の情報交換はファイルを直接アクセスさせるのではなく、ファイル内の情報を Content Provider や Service といったアプリ間連携の仕組みによって交換するべきである。これによりアプリ間のアクセス制御も実現できる。

SD カード等の外部記憶デバイスは十分なアクセス制御ができないため、容量の大きなファイルを扱う場合や別の場所 (PC など) への情報の移動目的など、機能上どうしても必要な場合のみに使用を限定するべきである。基本的に外部記憶デバイス上にはセンシティブな情報を含んだファイルを配置してはならない。もしセンシティブな情報を外部記憶デバイス上のファイルに保存しなければならない場合は暗号化等の対策が必要になるが、ここでは言及しない。

4.6.1 サンプルコード

前述のようにファイルは原則非公開にするべきである。しかしながらさまざまな事情によって、他のアプリにファイルを直接読み書きさせるべきときもある。セキュリティの観点から分類したファイルの種類と比較を 表 4.6.1 に示す。ファイルの格納場所や他アプリへのアクセス許可の組み合わせにより 4 種類のファイルに分類している。以降ではこのファイルの分類ごとにサンプルコードを示し説明を加えていく。

表 4.6.1: セキュリティ観点によるファイルの分類と比較

ファイルの分類	他アプリへのアクセス許可	格納場所	概要
非公開ファイル	なし	アプリディレクトリ内	• アプリ内でのみ読み書きできる。 • センシティブな情報を扱うことができる。 • ファイルは原則このタイプにするべき。
読み取り公開ファイル	読み取り	アプリディレクトリ内	• 他アプリおよびユーザーも読み取り可能。 • アプリ外部に公開（閲覧）可能な情報を扱う。 • 公開ファイルを作成するための MODE_WORLD_READABLE 変数は API Level 17 以降では非推奨で API Level 24 以降ではセキュリティ例外が発生する。
読み書き公開ファイル	読み取り・書き込み	アプリディレクトリ内	• 他アプリおよびユーザーも読み書き可能。 • セキュリティの観点からもアプリ設計の観点からも使用は避けるべき。
外部記憶ファイル	読み取り・書き込み	SD カードなどの外部記憶装置	• アクセス権のコントロールができない。 • 他アプリやユーザーによるファイルの読み書き・削除が常に可能。 • 使用は必要最小限にするべき。 • 比較的容量の大きなファイルを扱うことができる。 • API Level 29 以降は限定ビューを使用する。

次のページに続く

表 4.6.1 – 前のページからの続き

ファイルの分類	他アプリへのアクセス許可	格納場所	概要
外部記憶ファイル (API Level 29 以降)	読み取り・書き込み	SD カードなどの外部記憶装置	• 外部ストレージ用の限定ビューによりアプリ固有ディレクトリにそのアプリ用のファイルを保存できる。 • 他のアプリが作成したファイルへのアクセスには – アプリに READ_EXTERNAL_STORAGE パーミッションが付与されている – ファイルが MediaStore.Images、MediaStore.Video、MediaStore.Audio のいずれかに保存されている の両方の条件が必要。 • 他のファイル (ダウンロードディレクトリ内を含む) にアクセスするにはストレージアクセスフレームワークの使用が必要。 • 比較的容量の大きなファイルを扱うことができる。

4.6.1.1 非公開ファイルを扱う

同一アプリ内でのみ読み書きされるファイルを扱う場合であり、安全なファイルの使い方である。ファイルに格納する情報が公開可能かどうかに関わらず、できるだけファイルは非公開の状態で保持し、他アプリとの必要な情報のやり取りは別の Android の仕組み（Content Provider、Service) を利用して行うことを原則とする。

ポイント：

1. ファイルは、アプリディレクトリ内に作成する
2. ファイルのアクセス権は、他のアプリが利用できないようにプライベートモードにする
3. センシティブな情報を格納することができる
4. ファイルに格納する (された) 情報に対しては、その入手先に関わらず内容の安全性を確認する

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.
 * 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
```

(次のページに続く)

(前のページからの続き)

```
 * 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);
    }

    /**
     * ファイルの作成処理
     *
     * @param view
     */
    public void onCreateFileClick(View view) {
        FileOutputStream fos = null;
        try {
            // ★ポイント1★ ファイルは、アプリディレクトリ内に作成する
            // ★ポイント2★ ファイルのアクセス権は、他のアプリが利用できないようにプライベートモードにする
            fos = openFileOutput(FILE_NAME, MODE_PRIVATE);

            // ★ポイント3★ センシティブな情報を格納することができる
            // ★ポイント4★ ファイルに格納する情報に対しては、その入手先に関わらず内容の安全性を確認す
```

(次のページに続く)

(前のページからの続き)

```
る
            // サンプルにつき割愛。「3.2 入力データの安全性を確認する」を参照。
            fos.write(new String("センシティブな情報 (File Activity)\n").getBytes());
        } catch (FileNotFoundException e) {
            mFileView.setText(R.string.file_view);
        } catch (IOException e) {
            android.util.Log.e("PrivateFileActivity", "ファイルの作成に失敗しました");
        } finally {
            if (fos != null) {
                try {
                    fos.close();
                } catch (IOException e) {
                    android.util.Log.e("PrivateFileActivity", "ファイルの終了に失敗しました");
                }
            }
        }

        finish();
    }

    /**
     * ファイルの読み込み処理
     *
     * @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", "ファイルの読込に失敗しました");
        } finally {
            if (fis != null) {
                try {
                    fis.close();
```

(次のページに続く)

(前のページからの続き)

```
                } catch (IOException e) {
                    android.util.Log.e("PrivateFileActivity", "ファイルの終了に失敗しました");
                }
            }
        }
    }

    /**
     * ファイルの削除処理
     *
     * @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;
```

(次のページに続く)

(前のページからの続き)

```
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);
    }

    private void callFileActivity() {
        Intent intent = new Intent();
        intent.setClass(this, PrivateFileActivity.class);

        startActivity(intent);
    }

    /**
     * ファイル Activity の呼び出し処理
     *
     * @param view
     */
    public void onCallFileActivityClick(View view) {
        callFileActivity();
    }

    /**
     * ファイルの読み込み処理
     *
     * @param view
     */
    public void onReadFileClick(View view) {
        FileInputStream fis = null;
        try {
            fis = openFileInput(FILE_NAME);
```

(次のページに続く)

(前のページからの続き)

```
            byte[] data = new byte[(int) fis.getChannel().size()];

            fis.read(data);

            // ★ポイント 4★ ファイルに格納された情報に対しては、その入手先に関わらず内容の安全性を確認
する
            // サンプルにつき割愛。「3.2 入力データの安全性を確認する」を参照。
            String str = new String(data);

            mFileView.setText(str);
        } catch (FileNotFoundException e) {
            mFileView.setText(R.string.file_view);
        } catch (IOException e) {
            android.util.Log.d("PrivateFileActivity", "ファイルの読込に失敗しました");
        } finally {
            if (fis != null) {
                try {
                    fis.close();
                } catch (IOException e) {
                    android.util.Log.d("PrivateFileActivity", "ファイルの終了に失敗しました");
                }
            }
        }
    }

    /**
     * ファイルの追記処理
     *
     * @param view
     */
    public void onWriteFileClick(View view) {
        FileOutputStream fos = null;
        try {
            // ★ポイント 1★ ファイルは、アプリケーションディレクトリ内に作成する
            // ★ポイント 2★ ファイルのアクセス権は、他のアプリが利用できないようにプライベートモードに
する
            fos = openFileOutput(FILE_NAME, MODE_APPEND);

            // ★ポイント 3★ センシティブな情報を格納することができる
            // ★ポイント 4★ ファイルに格納する情報に対しては、その入手先に関わらず内容の安全性を確認す
る
            // サンプルにつき割愛。「3.2 入力データの安全性を確認する」を参照。
            fos.write(new String("センシティブな情報 (User Activity)\n").getBytes());
        } catch (FileNotFoundException e) {
```

(次のページに続く)

(前のページからの続き)

```
            mFileView.setText(R.string.file_view);
        } catch (IOException e) {
            android.util.Log.d("PrivateFileActivity", "ファイルの作成に失敗しました");
        } finally {
            if (fos != null) {
                try {
                    fos.close();
                } catch (IOException e) {
                    android.util.Log.d("PrivateFileActivity", "ファイルの終了に失敗しました");
                }
            }
        }

        callFileActivity();
    }

}
```

4.6.1.2 読み取り公開ファイルを扱う

不特定多数のアプリに対して内容を公開するためのファイルである。以下のポイントに気を付けて実装すれば、比較的安全なファイルの使い方になる。ただし、公開ファイルを作成するための、MODE_WORLD_READABLE 変数は API Level17 以降では deprecated となっており、API Level 24 以降ではセキュリティ例外が発生するため、以下のサンプルコードは動作しない。Content Provider によるファイル共有方法が望ましい。

ポイント：

1. ファイルは、アプリディレクトリ内に作成する
2. ファイルのアクセス権は、他のアプリに対しては読み取り専用モードにする
3. センシティブな情報は格納しない
4. ファイルに格納する (された) 情報に対しては、その入手先に関わらず内容の安全性を確認する

PublicFileActivity.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,
```

(次のページに続く)

(前のページからの続き)

```
 * 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);
    }

    /**
     * ファイルの作成処理
     *
     * @param view
     */
    public void onCreateFileClick(View view) {
        FileOutputStream fos = null;
        try {
            // ★ポイント1★ ファイルは、アプリディレクトリ内に作成する
            // ★ポイント2★ ファイルのアクセス権は、他のアプリに対しては読み取り専用モードにする
            //  （読み取り専用モードの MODE_WORLDREADABLE は API LEVEL 17 で deplicated となったため、
            //  極力使用せず、ContentProvider などによるデータのやり取りをすること）
            fos = openFileOutput(FILE_NAME, MODE_WORLD_READABLE);
```

(次のページに続く)

(前のページからの続き)

```
            // ★ポイント 3★ センシティブな情報は格納しない
            // ★ポイント 4★ ファイルに格納する (された) 情報に対しては、その入手先に関わらず内容の安全
性を確認する
            // サンプルにつき割愛。「3.2 入力データの安全性を確認する」を参照。
            fos.write(new String("センシティブでない情報 (Public File Activity)\n")
                    .getBytes());
        } catch (FileNotFoundException e) {
            mFileView.setText(R.string.file_view);
        } catch (IOException e) {
            android.util.Log.e("PublicFileActivity", "ファイルの作成に失敗しました");
        } finally {
            if (fos != null) {
                try {
                    fos.close();
                } catch (IOException e) {
                    android.util.Log.e("PublicFileActivity", "ファイルの終了に失敗しました");
                }
            }
        }

        finish();
    }

    /**
     * ファイルの読み込み処理
     *
     * @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", "ファイルの読込に失敗しました");
```

(次のページに続く)

(前のページからの続き)

```
        } finally {
            if (fis != null) {
                try {
                    fis.close();
                } catch (IOException e) {
                    android.util.Log.e("PublicFileActivity", "ファイルの終了に失敗しました");
                }
            }
        }
    }

    /**
     * ファイルの削除処理
     *
     * @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;
```

(次のページに続く)

(前のページからの続き)

```
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 {

    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 がありませんでした)");
        }
    }

    /**
```

(次のページに続く)

(前のページからの続き)

```
     * ファイル Activity の呼び出し処理
     *
     * @param view
     */
    public void onCallFileActivityClick(View view) {
        callFileActivity();
    }

    /**
     * ファイルの読み込み処理
     *
     * @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);

            // ★ポイント 4★ ファイルに格納する（された）情報に対しては、その入手先に関わらず内容の安全
性を確認する
            // サンプルにつき割愛。「3.2 入力データの安全性を確認する」を参照。
            String str = new String(data);

            mFileView.setText(str);
        } catch (FileNotFoundException e) {
            android.util.Log.e("PublicUserActivity", "ファイルがありません");
        } catch (IOException e) {
            android.util.Log.e("PublicUserActivity", "ファイルの読込に失敗しました");
        } finally {
            if (fis != null) {
                try {
                    fis.close();
                } catch (IOException e) {
                    android.util.Log.e("PublicUserActivity", "ファイルの終了に失敗しました");
                }
            }
        }
    }

    /**
```

(次のページに続く)

(前のページからの続き)

```
     * ファイルの追記処理
     *
     * @param view
     */
    public void onWriteFileClick(View view) {
        FileOutputStream fos = null;
        boolean exception = false;
        try {
            File file = new File(getFilesPath(FILE_NAME));
            // 書き込みは失敗する。FileNotFoundException が発生
            fos = new FileOutputStream(file, true);

            fos.write(new String("センシティブでない情報 (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);
            File file = new File(ctx.getFilesDir(), filename);
            path = file.getPath();
        } catch (NameNotFoundException e) {
            android.util.Log.e("PublicUserActivity", "ファイルがありません");
        }
        return path;
    }
}
```

4.6.1.3 読み書き公開ファイルを扱う

不特定多数のアプリに対して、読み書き権限を許可するファイルの使い方である。

不特定多数のアプリが読み書き可能ということは、マルウェアも当然内容の書き換えが可能であり、データの信頼性も安全性も全く保証されない。また、悪意のない場合でもファイル内のデータの形式や書き込みを行うタイミングなど制御が困難であり、そのようなファイルは機能面からも実用性が無いに等しい。

以上のように、セキュリティの観点からもアプリ設計の観点からも、読み書き公開ファイルを安全に運用することは不可能であり、読み書き公開ファイルの使用は避けなければならない。

ポイント：

1. 他アプリから読み書き可能なアクセス権を設定したファイルは作らない

4.6.1.4 外部記憶 (読み書き公開) ファイルを扱う

SD カードのような外部記憶デバイス上にファイルを格納する場合である。比較的容量の大きな情報を格納する (Web からダウンロードしたファイルを置くなどの) 場合や外部に情報を持ち出す (バックアップなどの) 場合に利用することが想定される。

「外部記憶 (読み書き公開) ファイル」は不特定多数のアプリに対して「読み取り公開ファイル」と同等の性質を持つ。さらに android.permission.WRITE_EXTERNAL_STORAGE Permission を利用宣言している不特定多数のアプリに対しては「読み書き公開ファイル」と同等の性質を持つ。そのため、外部記憶 (読み書き公開) ファイルの使用は必要最小限にとどめるべきである。

Android アプリの慣例として、バックアップファイルは外部記憶デバイス上に作成されることが多い。しかし外部記憶デバイス上のファイルは前述のようにマルウェアを含む他のアプリから改ざんや削除されてしまうリスクがある。ゆえにバックアップを出力するアプリでは「バックアップファイルは速やかに PC 等の安全な場所にコピーしてください」といった警告表示をするなど、アプリの仕様や設計面でのリスク最小化の工夫も必要となる。

なお、Android 10(API Level 29) では、対象範囲別ストレージ機能（「4.6.3.5. *Android 10* 以降の外部ストレージアクセス」参照）により、以下のサンプルコード（利用側）は動作しない。

ただし、マニフェスト属性 requestLegacyExternalStorage の設定により、対象範囲別ストレージ機能を一時的に無効にすることができる。これはアプリの完全互換性を確保する前やアプリのテストを行う前の一時的な用途に限って使用するものであり、リリース版などで使用してはならない。

それまで暫定的に用意されたマニフェスト属性 requestLegacyExternalStorage の設定は無視される。アプリ専用のディレクトリは外部に非公開となり、別のどのアプリからもアクセスできない。Android 11 をターゲットとする場合、対象範囲別ストレージと互換性のあるディレクトリにデータを移行する必要がある。ほとんどの場合、アプリ専用のディレクトリにデータを移行できる。

ポイント：

1. センシティブな情報は格納しない
2. アプリ毎にユニークなディレクトリにファイルを配置する
3. ファイルに格納する (された) 情報に対しては、その入手先に関わらず内容の安全性を確認する
4. 利用側のアプリで書き込みを行わない仕様にする

作成側のサンプルコード

AndroidManifest.xml

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

  <!-- android.permission.WRITE_EXTERNAL_STORAGE Permission を利用宣言する -->
  <!-- Android 4.4 (API Level 19) 以降では、外部ストレージのアプリデータ領域を
       読み書きする際に Permission が不要なため、maxSdkVersion を宣言する -->
  <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.
 */
```

(次のページに続く)

(前のページからの続き)

```
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;

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);
    }

    /**
     * ファイルの作成処理
     *
     * @param view
     */
    public void onCreateFileClick(View view) {
        FileOutputStream fos = null;
        try {
            // ★ポイント1★ センシティブな情報は格納しない
            // ★ポイント2★ アプリ毎にユニークなディレクトリにファイルを配置する
            File file = new File(getExternalFilesDir(TARGET_TYPE), FILE_NAME);
            fos = new FileOutputStream(file, false);

            // ★ポイント3★ ファイルに格納する(された)情報に対しては、その入手先に関わらず内容の安全
性を確認する
            // サンプルにつき割愛。「3.2 入力データの安全性を確認する」を参照。
```

(次のページに続く)

(前のページからの続き)

```
            fos.write(new String("センシティブでない情報 (External File Activity)\n")
                    .getBytes());
        } catch (FileNotFoundException e) {
            mFileView.setText(R.string.file_view);
        } catch (IOException e) {
            android.util.Log.e("ExternalFileActivity", "ファイルの読込に失敗しました");
        } finally {
            if (fos != null) {
                try {
                    fos.close();
                } catch (IOException e) {
                    android.util.Log.e("ExternalUserActivity", "ファイルの終了に失敗しました");
                }
            }
        }

        finish();
    }

    /**
     * ファイルの読み込み処理
     *
     * @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);

            // ★ポイント 3★ ファイルに格納する (された) 情報に対しては、その入手先に関わらず内容の安全
性を確認する
            // サンプルにつき割愛。「3.2 入力データの安全性を確認する」を参照。
            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", "ファイルの読込に失敗しました");
        } finally {
```

(次のページに続く)

(前のページからの続き)

```
            if (fis != null) {
                try {
                    fis.close();
                } catch (IOException e) {
                    android.util.Log.e("ExternalFileActivity", "ファイルの終了に失敗しました");
                }
            }
        }
    }

    /**
     * ファイルの削除処理
     *
     * @param view
     */
    public void onDeleteFileClick(View view) {

        File file = new File(getExternalFilesDir(TARGET_TYPE), FILE_NAME);
        file.delete();

        mFileView.setText(R.string.file_view);
    }
}
```

利用側のサンプルコード

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>

  <!-- Android 4.0.3 (API Level 14) 以降では、外部ストレージを読むための Permission が
       定義されたので利用宣言をする。実際には Android 4.4(API Level 19) 以降で
       他のアプリデータ領域を読む場合に必須となる -->
  <uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE"/>

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

(次のページに続く)

(前のページからの続き)

```
    <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.
 */

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;
```

(次のページに続く)

(前のページからの続き)

```
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 レベル 23）以降、アプリに dangerous パーミッション
         // （この場合 READ_EXTERNAL_STORAGE パーミッション）が必要な場合は、
         // ダイアログを表示し、ユーザの承認を得る。
         // （「5.2.3.6. Android 6.0 以降の Permission モデルの仕様変更について」参照）
        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
    public void onRequestPermissionsResult(int requestCode,
                                           String permissions[], int[] grantResults) {
        if (requestCode == MY_PERMISSIONS_REQUEST_READ_EXTERNAL_STORAGE) {
            if (grantResults[0] != PackageManager.PERMISSION_GRANTED) {
                finish();
            }
```

(次のページに続く)

(前のページからの続き)

```
        }
    }

    private void callFileActivity() {
        Intent intent = new Intent();
        intent.setClassName(TARGET_PACKAGE, TARGET_CLASS);

        try {
            startActivity(intent);
        } catch (ActivityNotFoundException e) {
            mFileView.setText("(File Activity がありませんでした)");
        }
    }

    /**
     * ファイル Activity の呼び出し処理
     *
     * @param view
     */
    public void onCallFileActivityClick(View view) {
        callFileActivity();
    }

    /**
     * ファイルの読み込み処理
     *
     * @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);

            // ★ポイント 3★ ファイルに格納する (された) 情報に対しては、その入手先に関わらず内容の安全性を確認する
            // サンプルにつき割愛。「3.2 入力データの安全性を確認する」を参照。
            String str = new String(data);

            mFileView.setText(str);
        } catch (FileNotFoundException e) {
```

(次のページに続く)

(前のページからの続き)

```
            mFileView.setText(R.string.file_view);
        } catch (IOException e) {
            android.util.Log.e("ExternalUserActivity", "ファイルの読込に失敗しました");
        } finally {
            if (fis != null) {
                try {
                    fis.close();
                } catch (IOException e) {
                    android.util.Log.e("ExternalUserActivity", "ファイルの終了に失敗しました");
                }
            }
        }
    }

    /**
     * ファイルの追記処理
     *
     * @param view
     */
    public void onWriteFileClick(View view) {

        // ★ポイント 4★ 利用側のアプリで書き込みを行わない仕様にする
        // ただし、悪意のあるアプリが上書き・削除などを行うことを想定してアプリの設計を行うこと

        final AlertDialog.Builder alertDialogBuilder =
                new AlertDialog.Builder(this);
        alertDialogBuilder.setTitle("ポイント 4");
        alertDialogBuilder.setMessage("利用側のアプリで書き込みを行わないこと");
        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 {
```

(次のページに続く)

(前のページからの続き)

```
            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", "ファイルがありません");
        }
        return path;
    }
}
```

4.6.2 ルールブック

ファイルを扱う場合には以下のルールを守ること。

1. ファイルは原則非公開ファイルとして作成する（必須）
2. 他のアプリから読み書き権限でアクセス可能なファイルは作成しない（必須）
3. *SD* カードなど外部記憶デバイスに格納するファイルの利用は必要最小限にする（必須）
4. ファイルの生存期間を考慮してアプリの設計を行う（必須）

4.6.2.1 ファイルは原則非公開ファイルとして作成する（必須）

「4.6. ファイルを扱う」「4.6.1.1. 非公開ファイルを扱う」で述べたように、格納する情報の内容に関わらずファイルは原則非公開にするべきである。Android のセキュリティ設計の観点からも、情報のやり取りとそのアクセス制御は Content Provider や Service などの Android の仕組みの中で行うべきであり、できない事情がある場合のみファイルのアクセス権で代用することを検討することになる。

各ファイルタイプのサンプルコードや以下のルールの項も参照のこと。

4.6.2.2 他のアプリから読み書き権限でアクセス可能なファイルは作成しない（必須）

「4.6.1.3. 読み書き公開ファイルを扱う」で述べたように、他のアプリに対してファイルの読み書きを許可すると、ファイルに格納される情報の制御ができない。そのため、セキュリティ的な観点からも機能・設計的な観点からも読み書き公開ファイルを利用した情報の共有を考えるべきではない。

4.6.2.3 SD カードなど外部記憶デバイスに格納するファイルの利用は必要最小限にする（必須）

「4.6.1.4. 外部記憶 *(*読み書き公開*)* ファイルを扱う」で述べたように、SD カードをはじめとする外部記憶デバイスにファイルを置くことは、セキュリティおよび機能の両方の観点から潜在的な問題を抱えることに繋がる。一方で、SD カードはアプリディレクトリより生存期間の長いファイルを扱え、アプリ外部にデータを持ち出すのに常時使える唯一のストレージなので、アプリの仕様によっては使用せざるを得ないケースも多いと考えられる。

外部記憶デバイスにファイルを格納する場合、不特定多数のアプリおよびユーザーが読み・書き・削除できることを考慮して、サンプルコードで述べたポイントを含めて以下のようなポイントに気をつけてアプリの設計を行う必要がある。

- 原則としてセンシティブな情報は外部記憶デバイス上のファイルに保存しない
- もしセンシティブな情報を外部記憶デバイス上のファイルに保存する場合は暗号化する
- 他アプリやユーザーに改ざんされては困る情報を外部記憶デバイス上のファイルに保存する場合は電子署名も一緒に保存する
- 外部記憶デバイス上のファイルを読み込む場合、読み込むデータの安全性を確認してからデータを利用する
- 他のアプリやユーザーによって外部記憶デバイス上のファイルはいつでも削除されることを想定してアプリを設計しなければならない

「4.6.2.4. ファイルの生存期間を考慮してアプリの設計を行う（必須）」も参照すること。

4.6.2.4 ファイルの生存期間を考慮してアプリの設計を行う（必須）

アプリディレクトリに保存されたデータは以下のユーザー操作により消去される。アプリの生存期間と一致する、またはアプリの生存期間より短いのが特徴である。

- アプリのアンインストール
- 各アプリのデータおよびキャッシュの消去（「設定」→「アプリケーション」→「アプリケーションの管理」）

SD カード等の外部記憶デバイス上に保存されたファイルは、アプリの生存期間よりファイルの生存期間が長いことが特徴である。さらに次の状況も想定する必要がある。

- ユーザーによるファイルの消去
- SD カードの抜き取り・差し替え・アンマウント
- マルウェアによるファイルの消去

このようにファイルの保存場所によってファイルの生存期間が異なるため、本節で説明したようなセンシティブな情報を保護する観点だけでなく、アプリとして正しい動作を実現する観点でもファイルの保存場所を正しく選択する必要がある。

4.6.3 アドバンスト

4.6.3.1 ファイルディスクリプタ経由のファイル共有

他のアプリに公開ファイルを直接アクセスさせるのではなく、ファイルディスクリプタ経由でファイル共有する方法がある。Content Provider と Service でこの方法が使える。Content Provider や Service の中で非公開ファイルをオープンし、そのファイルディスクリプタを相手のアプリに渡す。相手アプリはファイルディスクリプタ経由でファイルを読み書きできる。

他のアプリにファイルを直接アクセスさせるファイル共有方法とファイルディスクリプタ経由のファイル共有方法の比較を 表 4.6.2 に示す。アクセス権のバリエーションとアクセス許可するアプリの範囲でメリットがある。特にアクセスを許可するアプリを細かく制御できるところがセキュリティ観点ではメリットが大きい。

表 4.6.2　アプリ間ファイル共有方法の比較

ファイル共有方法	アクセス権設定のバリエーション	アクセスを許可するアプリの範囲
他のアプリにファイルを直接アクセスさせるファイル共有	• 読み取り • 書き込み • 読み取り＋書き込み	すべてのアプリに対して一律アクセス許可してしまう
ファイルディスクリプタ経由のファイル共有	• 読み取り • 書き込み • 追記のみ • 読み取り＋書き込み • 読み取り＋追記のみ	Content Provider や Service にアクセスしてくるアプリに対して個別に一時的にアクセス許可・不許可を制御できる

上記ファイル共有方法のどちらにも共通することであるが、他のアプリにファイルの書き込みを許可するとファイル内容の完全性が保証しづらくなる。特に複数のアプリから同時に書き込みが行われると、ファイル内容のデータ構造が壊れてしまいアプリが正常に動作しなくなるリスクがある。他のアプリとのファイル共有においては、読み込み権限だけを許可するのが望ましい。

以下では、Content Provider でファイルを共有する実装例 (非公開 Provider の場合) をサンプルコードとして掲載する。

ポイント

1. 利用元アプリは自社アプリであるから、センシティブな情報を保存してよい
2. 自社限定 Content Provider アプリからの結果であっても、結果データの安全性を確認する

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.file.inhouseprovider;

import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.IOException;
```

(次のページに続く)

(前のページからの続き)

```
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";

    // 自社の Signature Permission
    private static final String MY_PERMISSION =
        "org.jssec.android.file.inhouseprovider.MY_PERMISSION";

    // 自社の証明書のハッシュ値
    private static String sMyCertHash = null;

    private static String myCertHash(Context context) {
        if (sMyCertHash == null) {
            if (Utils.isDebuggable(context)) {
                // debug.keystore の"androiddebugkey"の証明書ハッシュ値
                sMyCertHash = "0EFB7236 328348A9 89718BAD DF57F544 D5CCB4AE B9DB34BC␣
↪1E29DD26 F77C8255";
            } else {
                // keystore の"my company key"の証明書ハッシュ値
                sMyCertHash = "D397D343 A5CBC10F 4EDDEB7C A10062DE 5690984F 1FB9E88B␣
↪D7B3A7C2 42E142CA";
            }
        }
        return sMyCertHash;
    }

    @Override
    public boolean onCreate() {
        File dir = getContext().getFilesDir();
        FileOutputStream fos = null;
        try {
            fos = new FileOutputStream(new File(dir, FILENAME));
            // ★ポイント 1★ 利用元アプリは自社アプリであるから、センシティブな情報を保存してよい
            fos.write(new String("センシティブな情報").getBytes());
```

(次のページに続く)

(前のページからの続き)

```
        } catch (IOException e) {
            android.util.Log.e("InHouseProvider", "ファイル保存に失敗しました");
        } finally {
            try {
                fos.close();
            } catch (IOException e) {
                android.util.Log.e("InHouseProvider", "ファイル終了に失敗しました");
            }
        }

        return true;
    }

    @Override
    public ParcelFileDescriptor openFile(Uri uri, String mode)
        throws FileNotFoundException {

        // 独自定義 Signature Permission が自社アプリにより定義されていることを確認する
        if (!SigPerm
            .test(getContext(), MY_PERMISSION, myCertHash(getContext()))) {
            throw new SecurityException(
                "独自定義 Signature Permission が自社アプリにより定義されていない。");
        }

        File dir = getContext().getFilesDir();
        File file = new File(dir, FILENAME);

        // サンプルのため読み取り専用を常に返す
        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
```

(次のページに続く)

(前のページからの続き)

```
    public Uri insert(Uri uri, ContentValues values) {
        return null;
    }

    @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;
```

(次のページに続く)

(前のページからの続き)

```
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 {

    // 利用先の Content Provider 情報
    private static final String AUTHORITY = "org.jssec.android.file.inhouseprovider";

    // 自社の Signature Permission
    private static final String MY_PERMISSION =
        "org.jssec.android.file.inhouseprovider.MY_PERMISSION";

    // 自社の証明書のハッシュ値
    private static String sMyCertHash = null;

    private static String myCertHash(Context context) {
        if (sMyCertHash == null) {
            if (Utils.isDebuggable(context)) {
                // debug.keystore の"androiddebugkey"の証明書ハッシュ値
                sMyCertHash = "0EFB7236 328348A9 89718BAD DF57F544 D5CCB4AE B9DB34BC␣
↪1E29DD26 F77C8255";
            } else {
                // keystore の"my company key"の証明書ハッシュ値
                sMyCertHash = "D397D343 A5CBC10F 4EDDEB7C A10062DE 5690984F 1FB9E88B␣
↪D7B3A7C2 42E142CA";
            }
        }
        return sMyCertHash;
    }

    // 利用先 Content Provider のパッケージ名を取得
    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;
    }
```

(次のページに続く)

(前のページからの続き)

```
    public void onReadFileClick(View view) {

        logLine("[ReadFile]");

        // 独自定義 Signature Permission が自社アプリにより定義されていることを確認する
        if (!SigPerm.test(this, MY_PERMISSION, myCertHash(this))) {
            logLine("  独自定義 Signature Permission が自社アプリにより定義されていない。");
            return;
        }

        // 利用先 Content Provider アプリの証明書が自社の証明書であることを確認する
        String pkgname = providerPkgname(this, AUTHORITY);
        if (!PkgCert.test(this, pkgname, myCertHash(this))) {
            logLine("  利用先 Content Provider は自社アプリではない。");
            return;
        }

        // 自社限定 Content Provider アプリに開示してよい情報に限りリクエストに含めてよい
        ParcelFileDescriptor pfd = null;
        try {
            pfd = getContentResolver()
                  .openFileDescriptor(Uri.parse("content://" + AUTHORITY), "r");
        } catch (FileNotFoundException e) {
            android.util.Log.e("InHouseUserActivity", "ファイルがありません");
        }

        if (pfd != null) {
            FileInputStream fis = new FileInputStream(pfd.getFileDescriptor());

            if (fis != null) {
                try {
                    byte[] buf = new byte[(int) fis.getChannel().size()];
                    fis.read(buf);
                    // ★ポイント 2★ 自社限定 Content Provider アプリからの結果であっても、結果データの安全性を確認する
                    // サンプルにつき割愛。「3.2 入力データの安全性を確認する」を参照。
                    logLine(new String(buf));
                } catch (IOException e) {
                    android.util.Log.e("InHouseUserActivity", "ファイルの読み込みに失敗しました");
                } finally {
                    try {
                        fis.close();
                    } catch (IOException e) {
```

(次のページに続く)

(前のページからの続き)

```
                        android.util.Log.e("InHouseUserActivity", "ファイルの終了に失敗しました
↪");
                    }
                }
            }
            try {
                pfd.close();
            } catch (IOException e) {
                android.util.Log.e("InHouseUserActivity", "ファイルディスクリプタの終了に失敗しま
した");
            }

        } 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);
        mLogView.append("\n");
    }
}
```

4.6.3.2 ディレクトリのアクセス権設定

これまでファイルに着目してセキュリティの考慮点を説明してきた。ファイルのコンテナであるディレクトリについてもセキュリティの考慮が必要である。ここではディレクトリのアクセス権設定についてセキュリティ上の考慮ポイントを説明する。

Android には、アプリディレクトリ内にサブディレクトリを取得・作成するメソッドがいくつか用意されている。主なものを 表 4.6.3 に示す。

表 4.6.3　アプリディレクトリ配下のサブディレクトリ取得・作成メソッド

	他アプリに対するアクセス権の指定	ユーザーによる削除
Context#getFilesDir()	不可 (実行権限のみ)	「設定」→「アプリ」→アプリケーションを選択→「データを消去」
Context#getCacheDir()	不可 (実行権限のみ)	「設定」→「アプリ」→アプリケーションを選択→「キャッシュを消去」(※「データを消去」でも削除される)
Context#getDir(String name, int mode)	mode として、MODE_PRIVATE、MODE_WORLD_READABLE、MODE_WORLD_WRITABLE を指定可能	「設定」→「アプリ」→アプリケーションを選択→「データを消去」

ここで特に気を付けるのは Context#getDir() によるアクセス権の設定である。ファイルの作成でも説明しているように、Android のセキュリティ設計の観点からディレクトリも基本的には非公開にするべきであり、アクセス権の設定によって情報の共有を行うと思わぬ副作用があるので、情報の共有には他の手段を考えるべきである。

MODE_WORLD_READABLE

すべてのアプリに対してディレクトリの読み取り権限を与えるフラグである。すべてのアプリがディレクトリ内のファイル一覧や個々のファイルの属性情報を取得可能になる。このディレクトリ配下に秘密のファイルを配置することはできないため、通常はこのフラグを使用してはならない[*23]。

MODE_WORLD_WRITEABLE

他アプリに対してディレクトリの書き込み権限を与えるフラグである。すべてのアプリがディレクトリ内のファイルを作成、移動[*24]、リネーム、削除が可能になる。これらの操作はファイル自体のアクセス権設定（読み取り、書き込み、実行）とは無関係であり、ディレクトリの書き込み権限があるだけで可能となる操作であることに注意が必要だ。他のアプリから勝手にファイルを削除されたり置き換えられたりするため、通常はこのフラグを使用してはならない[*23]。

表 4.6.3 の「ユーザーによる削除」に関しては、「4.6.2.4. ファイルの生存期間を考慮してアプリの設計を行う（必須）」を参照のこと。

4.6.3.3 Shared Preference やデータベースファイルのアクセス権設定

Shared Preference やデータベースもファイルで構成される。アクセス権設定についてはファイルと同じことが言える。したがって Shared Preference もデータベースもファイルと同様に基本的には非公開ファイルとして作成し、内容の共有は Android のアプリ間連携の仕組みによって実現するべきである。

Shared Preference の使用例を次に示す。MODE_PRIVATE により非公開ファイルとして Shared Preference を作成している。

Shared Preference ファイルにアクセス制限を設定する例

```
import android.content.SharedPreferences;
import android.content.SharedPreferences.Editor;
```

(次のページに続く)

[*23] MODE_WORLD_READABLE および MODE_WORLD_WRITEABLE は API Level17 以降では deprecated となっており、API Level 24 以降ではセキュリティ例外が発生するため使用できなくなっている。

[*24] 内部ストレージから外部記憶装置 (SD カードなど) への移動などマウントポイントを超えた移動はできない。そのため、読み取り権限のない内部ストレージファイルが外部記憶装置に移動されて読み書き可能になるようなことはない。

（前のページからの続き）

```
// ～省略～
// Shared Preference を取得する（なければ作成される）
// ポイント：基本的に MODE_PRIVATE モードを指定する
SharedPreferences preference = getSharedPreferences(PREFERENCE_FILE_NAME, MODE_PRIVATE);
// 値が文字列のプリファレンスを書き込む例
Editor editor = preference.edit();
editor.putString("prep_key", "prep_value");// key:"prep_key", value:"prep_value"
editor.commit();
```

データベースについては「4.5. *SQLite* を使う」を参照すること。

4.6.3.4 Android 10 以降の内部ストレージアクセス

内部ストレージに関しては Android 9 以前と仕様の変更はない。内部ストレージはもともとアプリごとに分離されており、他アプリのデータにはアクセスできない設計となっている。

自アプリの内部ストレージ領域（/data/data/<package_name>/files/ など）には、openFileOutput() や FileOutputStream などの Context のファイル I/O メソッドを利用して、パーミッション不要で自由に読み書きできる。一方、他アプリの内部ストレージ領域には、Android 9 以前も 10 以降も一切アクセスできない。内部ストレージのデータはアプリがアンインストールされると同時に削除される。

このように、内部ストレージのアクセス仕様は Android 9 と 10 以降で変化はなく、常に「自アプリのみがアクセス可能」「他アプリのデータはアクセス不可」という原則が維持されている。

表 4.6.4　Android 10 以降の内部ストレージアクセス

用途	アクセス方法	パスの取得方法	実際のパス	必要なパーミッション
自アプリのプライベートデータ	Context のファイル I/O メソッド File API	context.getFilesDir()	/data/data/<package_name>/files/	不要
他アプリのプライベートデータ	通常アクセス不可	N/A	N/A	N/A

4.6.3.5 Android 10 以降の外部ストレージアクセス

Android 10 以降では、外部ストレージ（SD カードを含む）へのアクセスに関するセキュリティとプライバシーが大幅に強化された。ユーザーのプライバシー保護を向上させるため、他アプリのデータへのアクセスが厳しく制限されるようになった。

Android 9 までは、外部ストレージ内の自アプリの固有データにはパーミッション不要でアクセスできた一方、他アプリの固有データや他アプリのセカンダリストレージにアクセスする場合は READ_EXTERNAL_STORAGE や WRITE_EXTERNAL_STORAGE パーミッションが必要だった。File API を使い、直接パスを指定して他アプリのデータにもアクセスできる仕様だった。

Android 10 以降は Scoped Storage の導入により、他アプリの固有データや他アプリの外部ストレージ領域へのアクセス

が基本的に禁止された。これにより、他アプリが保存したデータへのアクセスは大幅に制限され、File API 等を使って直接パスを指定してもアクセスできなくなった。自アプリの外部ストレージ領域（/storage/emulated/0/Android/data/<package_name>/files/ など）へのアクセス方法やパスの取得方法には変更はなく、引き続きパーミッション不要で自由に読み書きできる。

以下に、Android 9 と 10 以降の外部ストレージアクセスの違いをまとめる。

表 4.6.5　Android 9 の外部ストレージアクセス

用途	アクセス方法	パスの取得方法	実際のパス	必要なパーミッション
自アプリの固有のデータ	File API	context.getExternalFilesDir(null) context.getExternalFilesDirs(null)[1]	/storage/emulated/0/Android/data/<package_name>/files/ /storage/extSdCard/ /storage/sdcard1/	不要
他アプリの固有のデータ	File API	直接パス指定	/storage/emulated/0/Android/data/<other_package_name>/files/ /storage/extSdCard/ /storage/sdcard1/	READ_EXTERNAL_STORAGE WRITE_EXTERNAL_STORAGE

表 4.6.6　Android 10 以降の外部ストレージアクセス

用途	アクセス方法	パスの取得方法	実際のパス	必要なパーミッション
自アプリの固有のデータ	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/	不要
他アプリの固有のデータ	通常アクセス不可	N/A	N/A	N/A

4.6.3.6 Android 10 以降の共有ストレージアクセス

Android 10 以降では、共有ストレージへのアクセスに関するセキュリティとプライバシーの強化が行われている。そのため、従来のパスベースの File API による直接アクセスは推奨されず、MediaStore API や Storage Access Framework（SAF）を利用して間接的にファイルへアクセスすることが基本となっている。

自アプリが自身で保存・管理するファイルについては、MediaStore API や SAF を利用することで、特別な権限を要求せずに読み書きが可能である。これにより、アプリ固有のデータは自由に操作できる。一方で、他アプリが保存したファイル（たとえばユーザーの写真やダウンロードフォルダ内のファイル）にアクセスする場合は、読み取り権限（例：READ_MEDIA_IMAGES や READ_EXTERNAL_STORAGE など）が必要となる。なお、これら他アプリのデータについては、読み取りのみが許可されており、直接書き込みや削除はできない。

Android 13 以降では、ユーザーが端末内の画像や動画など既存のメディアファイルをアプリで利用する際、「ユーザーが選択した既存ファイルを読み取る」用途に限って OS 標準の「Photo Picker」機能が追加された。Photo Picker はストレージ権限が不要で、ユーザーが選択した画像や動画ファイルのみに一時的なアクセスができるため、プライバシー保護の観点から Google 公式により推奨されている。Photo Picker 経由で取得したファイルも、読み取り専用であり、書き込みはできない。

まとめると、Android 9 以前では File API やディレクトリのパスを直接使用してファイルにアクセスできたが、Android 10 以降はセキュリティ強化のため、MediaStore API や SAF を使用して間接的にファイルにアクセスすることが求められる。自アプリのファイルは権限不要で読み書きできるが、他アプリのファイルは権限が必要で読み取りのみ可能であり、書き込みはできない。Android 13 以降は、画像・動画などの既存メディアファイルを「ユーザーが選択してアプリに渡す」場合は Photo Picker の利用が推奨される。自アプリのファイル管理や新規保存には、引き続き MediaStore API や SAF を利用する。

表 4.6.7　Android 9 の共有ストレージアクセス

用途	アクセス方法	パスの取得方法	実際のパス	必要なパーミッション
自アプリの共有データ	File API	Environment.getExternalStoragePublicDirectory() Environment.DIRECTORY_MUSIC Environment.DIRECTORY_PICTURES Environment.DIRECTORY_MOVIES など	/storage/emulated/0/Music /storage/emulated/0/Pictures /storage/emulated/0/Movies など	READ_EXTERNAL_STORAGE WRITE_EXTERNAL_STORAGE
他アプリの共有データ	File API	Environment.getExternalStoragePublicDirectory() Environment.DIRECTORY_MUSIC Environment.DIRECTORY_PICTURES Environment.DIRECTORY_MOVIES など	/storage/emulated/0/Music /storage/emulated/0/Pictures /storage/emulated/0/Movies など	READ_EXTERNAL_STORAGE WRITE_EXTERNAL_STORAGE

表 4.6.8　Android 10 以降の共有ストレージアクセス

用途	アクセス方法	パスの取得方法	実際のパス	必要なパーミッション
自アプリの共有データ	MediaStore API Storage Access Framework	ContentResolver SAF	ContentResolver で取得した URI SAF で取得した URI	不要
他アプリの共有データ	MediaStore API Storage Access Framework	ContentResolver SAF	ContentResolver で取得した URI SAF で取得した URI	ストレージ読み取りパーミッション ユーザーの明示的な同意が必要

表 4.6.9　Android 13 以降の共有ストレージアクセス

用途	アクセス方法	パスの取得方法	実際のパス	必要なパーミッション
自アプリの共有データ	MediaStore API Storage Access Framework	ContentResolver SAF	ContentResolver で取得した URI SAF で取得した URI	不要
他アプリの共有データ	MediaStore API Storage Access Framework	ContentResolver SAF	ContentResolver で取得した URI SAF で取得した URI	ストレージ読み取りパーミッション ユーザーの明示的な同意が必要
既存の画像や動画ファイル	Photo Picker API	Photo Picker API	Photo Picker API で取得した URI	不要

自アプリの共有データ

自アプリの共有データのアクセス方法について、 MediaStore API を利用したサンプルコードを以下に示す。具体的には、Documents フォルダに「example.txt」という名前のテキストファイルを作成し、その中に「Hello,World!」という内容を書き込む。

```
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
```

(次のページに続く)

(前のページからの続き)

```
@RequiresApi(Build.VERSION_CODES.Q)
class MainActivity : AppCompatActivity() {

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        writeFileToMediaStore()
    }

    private fun writeFileToMediaStore() {
        // MediaStore に保存するための ContentValues を設定
        val contentValues = ContentValues().apply {
            put(MediaStore.MediaColumns.DISPLAY_NAME, "example.txt")
            put(MediaStore.MediaColumns.MIME_TYPE, "text/plain")
            put(MediaStore.MediaColumns.RELATIVE_PATH, "Documents/")
        }

        // MediaStore に新しい項目を挿入
        val uri = contentResolver.insert(MediaStore.Files.getContentUri("external"),␣
↪contentValues)

        // URI を使用してファイルにデータを書き込む
        uri?.let {
            val outputStream: OutputStream? = contentResolver.openOutputStream(it)
            outputStream?.use {
                it.write("Hello, World!".toByteArray())
            }
        }
    }
}
```

他アプリの共有データ

他アプリの共有データにアクセスする場合、MediaStore API を利用することで画像・動画・音楽・ドキュメント等のメディアファイルを読み取ることができる。ただし、この場合はストレージ関連の権限が必要となり、要求すべき権限の種類やその挙動は Android のバージョンごとに異なる。また、他アプリの共有データは「読み取りのみ」可能であり、直接書き込みや削除はできないことに注意が必要である。

以下に、MediaStore API を用いた他アプリの共有データアクセス時の権限要件と、そのバージョンごとの違いをまとめる。

表 4.6.10　Android 10 以降のストレージ読み取りパーミッション

メディアの種類	用途	アクセス方法	必要なパーミッション
画像	他アプリの画像データ	MediaStore.Images.Media.EXTERNAL_CONTENT_URI	READ_EXTERNAL_STORAGE
音楽	他アプリの音楽データ	MediaStore.Audio.Media.EXTERNAL_CONTENT_URI	READ_EXTERNAL_STORAGE
ビデオ	他アプリのビデオデータ	MediaStore.Video.Media.EXTERNAL_CONTENT_URI	READ_EXTERNAL_STORAGE
ドキュメント	他アプリのドキュメントデータ	MediaStore.Files.getContentUri(”external”)	READ_EXTERNAL_STORAGE

表 4.6.11　Android 13 以降のストレージ読み取りパーミッション

メディアの種類	用途	アクセス方法	必要なパーミッション
画像	他アプリの画像データ	MediaStore.Images.Media.EXTERNAL_CONTENT_URI	READ_MEDIA_IMAGES
音楽	他アプリの音楽データ	MediaStore.Audio.Media.EXTERNAL_CONTENT_URI	READ_MEDIA_AUDIO
ビデオ	他アプリのビデオデータ	MediaStore.Video.Media.EXTERNAL_CONTENT_URI	READ_MEDIA_VIDEO
ドキュメント	他アプリのドキュメントデータ	MediaStore.Files.getContentUri(”external”)	READ_EXTERNAL_STORAGE

表 4.6.12　Android 14 以降のストレージ読み取りパーミッション

メディアの種類	用途	アクセス方法	必要なパーミッション
画像	他アプリの画像データ	MediaStore.Images.Media.EXTERNAL_CONTENT_URI	READ_MEDIA_IMAGES
音楽	他アプリの音楽データ	MediaStore.Audio.Media.EXTERNAL_CONTENT_URI	READ_MEDIA_AUDIO
ビデオ	他アプリのビデオデータ	MediaStore.Video.Media.EXTERNAL_CONTENT_URI	READ_MEDIA_VIDEO
ドキュメント	他アプリのドキュメントデータ	MediaStore.Files.getContentUri(”external”)	READ_EXTERNAL_STORAGE
ユーザー選択のビジュアルメディア	ユーザーが選択した画像やビデオデータ	MediaStore.Images.Media.EXTERNAL_CONTENT_URI / MediaStore.Video.Media.EXTERNAL_CONTENT_URI	READ_MEDIA_VISUAL_USER_SELECTED

READ_MEDIA_VIDEO を使用したサンプルコードを以下に示す。デバイスに保存されているビデオファイルのタイトルを読み込み画面に表示する。

```
<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
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 {
            // パーミッションが拒否された場合の処理
        }
    }

    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)
                    )
```

(次のページに続く)

(前のページからの続き)

```
                }
            }
        }
        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) -> {
                    // ユーザーに説明を表示する
                }
                else -> {
                    requestPermissionLauncher.launch(android.Manifest.permission.READ_
↪MEDIA_VIDEO)
                }
            }
        } else {
            // 古いバージョンの Android デバイス向けのフォールバック
            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)
```

(次のページに続く)

(前のページからの続き)

```
                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"))
    }
}
```

既存の画像や動画ファイル

既存の画像や動画ファイルにアプリがアクセスする場合、Android 13 以降は Photo Picker の利用が推奨されている。Photo Picker はユーザーが端末やクラウド上から明示的に画像や動画を選択する仕組みで、ストレージ権限を要求しないのが特徴となる。これはユーザー自身がどのファイルをアプリに渡すかを毎回選択するためであり、アプリが自動的に全てのメディアファイルへアクセスできるわけではない。Photo Picker 経由で取得したファイルは一時的にのみ利用でき、恒久的なアクセス権が与えられるわけではない。Photo Picker で提供される機能は限定的であり、たとえば独自のギャラリー UI やファイルの継続的な管理など、Photo Picker で対応できない要件がある場合には MediaStore API や Storage Access Framework（SAF）の利用を検討することになる。

Photo Picker を使用したサンプルコードを以下に示す。「画像を選択」ボタンを押すと Photo Picker が起動し、選択した画像ファイルの URI が画面に表示される。

```
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
```

(次のページに続く)

(前のページからの続き)

```
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("画像を選択")
        }
```

(次のページに続く)

(前のページからの続き)

```
        if (selectedUri != null) {
            Text("選択された画像の URI: $selectedUri")
        }
    }
}

@Preview(showBackground = true)
@Composable
fun PhotoPickerDemoPreview() {
    MyApplicationTheme {
        PhotoPickerDemo()
    }
}
```

4.6.3.7 DCL (Dynamic Code Loading) の安全性の強化

Android 14 から、動的にコードを読み込む場合は、DEX、JAR、APK など読み込み対象となるファイルを開いた直後に、setReadOnly() メソッドを使用し読み取り専用にすることが必須となった。

これは、ファイルの競合状態を防ぐだけではなく、読み込むコード自体が改ざんされアプリが不正に使用されるリスクを軽減するためである。

setReadOnly() メソッドを使用した実装例は以下の通り。

```
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());
```

setReadOnly() メソッドを使用しなかった場合、PathClassLoader インスタンスを生成するタイミングで以下のようなエラーが発生する[*25] 。

```
E  Attempt to load writable dex file: /data/user/0/com.example.test/cache/Test_dex.jar
```

なお、セキュリティの観点からは、DCL を使用すること自体が推奨されていない。上述の通り、コードインジェクションやコード改ざんのリスクが考えられるためである。ユーザーは、アプリが内部でどのようなコードをロードしているかを意識することはない。アプリの挙動における責任はすべてアプリの提供側に帰属するものと考える。そのため、もし現状 DCL を使用しているのであれば、代替の方法を検討すべきだろう。

[*25] 公式では例外がスローされるとなっているが実際に検証したところエラーが発生する
https://developer.android.com/about/versions/14/behavior-changes-14?hl=ja#safer-dynamic-code-loading

公式では代替方法として Android App Bundle を使用することが推奨されており、2021 年 8 月から、Google Play ストアのすべての新しいアプリにおいて、Android App Bundle の使用が必須となっている[*26] 。

Android 17 におけるネイティブライブラリへの適用拡大

Android 17 以降ではこの DCL の保護がさらに強化され、これまで DEX や JAR 等に適用されていた読み取り専用の要件が、ネイティブライブラリ（.so ファイル）にも拡張された。具体的には、 System.load() などを利用して動的に読み込むネイティブファイルは、ロードされる時点でファイルが読み取り専用（read-only）としてマークされている必要がある。ファイルが書き込み可能な状態のまま読み込もうとすると、システムが UnsatisfiedLinkError をスローし実行がブロックされる。これにより、実行直前にネイティブコードが不正に改ざんされる攻撃領域が減少し、アプリの安全性がより向上している。

DCL の根本的なリスクと推奨される代替手段

そもそもセキュリティの観点から、DCL を使用すること自体が推奨されていない。コードの動的なロードは、コードインジェクションや改ざんによって機密データの窃取や有害な処理の実行に繋がるリスクがあるためである。ユーザーは、アプリが内部でどのようなコードをロードしているかを意識することはなく、アプリの挙動における責任はすべてアプリの提供側に帰属する。また、リモートソースからの DCL は Google Play のポリシー違反になり得る。

そのため、開発者はまずアプリや利用している SDK が System.load() などを用いていないか確認し、ビジネス上の合理的な理由がない限りは DCL を避けるべきである。公式でも代替方法として Android App Bundle を使用することが推奨されており、できるだけ機能を事前に APK や App Bundle に含める設計へ移行することが望ましい。

やむを得ず **DCL** を使用する場合のセキュリティ対策

アプリの要件上どうしてもネイティブライブラリ等を動的に読み込む必要がある場合は、改ざんリスクを抑えるために以下の対策を講じること。

1. アクセス制限された保存場所の利用: 動的ロードするコードの保存先として、外部から容易に触れることができる場所（外部ストレージなど）は避け、アプリの内部ストレージや Scoped Storage など、アクセスが厳格に制限された領域を使用する。
2. ファイルの読み取り専用化: 読み込む前にファイルが確実に読み取り専用になっていることを確認する。
3. 整合性チェックと署名検証: ロードする直前にファイルのハッシュ検証（SHA-256 など）を行い、ファイルが改ざんされていないことを確認する。ハッシュ値自体も安全な領域に保管し、可能であればコード署名とその検証も組み合わせることで、コードの真正性を確実なものにすることが推奨される。

4.6.3.8 メディア所有者のパッケージ名

MediaStore の MediaColumns は、TITLE や DISPLAY_NAME、MIME_TYPE など、ファイルに関する各種情報を列として保存しており、これらの列情報を条件にして端末内のメディアファイルをクエリすることができる。

Android 14 から、この列情報のうち、メディアファイルを作成したアプリのパッケージ名を表す OWNER_PACKAGE_NAME については、値が削除され、クエリ条件として指定することができなくなった。

これは、インストール済みアプリの情報がユーザーの個人情報および機密情報と見なされるようになったためである。

以下は、端末に保存されたオーディオファイルのクエリを行い、当該ファイルの OWNER_PACKAGE_NAME を表示するサンプルコードである。

[*26] 「アプリの中核品質」プライバシーとセキュリティ SC-E1
https://developer.android.com/docs/quality-guidelines/core-app-quality?hl=ja

```
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)));
```

このコードを Android 13(API Level 33) と Android 14(API Level 34) で実行した際の違いを以下に示す。

Android 13 で実行した場合:

```
D  voicerecorder.audiorecorder.voice
```

Android 14 で実行した場合:

```
E  Unknown message received from debugger! ''
```

Android 13 で実行した場合は、メディアファイルを作成したアプリのパッケージ名が出力されている一方、Android 14 では OWNER_PACKAGE_NAME によるクエリが行えず Logcat の出力にエラーメッセージが表示されていることがわかる。

4.6.3.9 Zip ファイルの Path Traversal 対策

Android 14 から、Zip ファイルのエントリに「..」が含まれる場合や「/」で始まる場合に、ZipException がスローされるようになった。

以下のような Zip ファイルである。

```
$ unzip -l test.zip
Archive:  test.zip
  Length      Date    Time    Name
---------  ---------- -----   ----
   104565  2023/02/28 14:21   ../mkcsv.txt
---------                     -------
   104565                     1 file
```

上記、test.zip のエントリは「../mkcsv.txt」となっている。このような Zip ファイルに、Android 14 をターゲットとしたアプリからアクセスした場合のサンプルコードと実行結果は以下の通りである。

```
filename = this.getFilesDir().getPath() + "/test.zip";

try {
    in = new ZipInputStream(new FileInputStream(filename));

    while ((zipEntry = in.getNextEntry()) != null) {
```

(次のページに続く)

(前のページからの続き)

```
    // 中略
} 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)
```

実行結果のログにもある通り ZipException がスローされており、これは getNextEntry() 実行時に発生する。

相対パスから意図しないパスにアクセスできてしまうパストラバーサル脆弱性への対策であり、開発者になんらかの対応を促すものである。

一時的にこの仕様をオプトアウトにするには、以下のように、dalvik.system.ZipPathValidator.clearCallback() を実行する。

```
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) {
    // 中略
} catch (IOException e) {
    e.printStackTrace();
}
```

4.6.3.10 選択した写真へのアクセスについて、直近のユーザーの選択をクエリする

Android 15 から、アプリはメディアアクセス権が部分的に付与された場合に、最近選択された写真やビデオのみを強調表示できるようになった。この機能により、頻繁に写真やビデオへのアクセスを要求するアプリのユーザー体験が向上する。この機能をアプリで利用するためには、ContentResolver を介して MediaStore にクエリを送信する際に、QUERY_ARG_LATEST_SELECTION_ONLY 引数を有効にする必要がある。

セキュリティ上の利点

この機能は、ユーザーのプライバシーとデータのセキュリティに対していくつかの重要な利点を提供する。

1. 限定されたアクセス範囲: QUERY_ARG_LATEST_SELECTION_ONLY 引数を使用することで、アプリはユーザーが最近選択した写真やビデオのみにアクセスすることができる。これにより、アプリがユーザーの全メディアライブラリにアクセスすることを防ぎ、不要なデータアクセスを制限する。

2. 最小限のデータ漏洩リスク: 部分的なアクセス権限を付与することで、必要最低限のデータに対してのみアクセスが許可される。これにより、アプリが意図せずに他の個人的な写真やビデオにアクセスするリスクを減少させる

ことができる。

3. ユーザーの安心感: ユーザーは、アプリが全てのメディアファイルにアクセスするのではなく、特定の写真やビデオのみにアクセスすることを知っているため、プライバシーがより保護されているという安心感を持つことができる。これにより、アプリの信頼性が向上し、ユーザーの利用意欲も高まる。

4. コンプライアンス対応: 近年、多くの国や地域でデータプライバシーに関する法規制が強化されている。例えば、GDPR（一般データ保護規則）や CCPA（カリフォルニア消費者プライバシー法）など。この機能を利用することで、アプリはこれらの法律や規制により適合しやすくなる。

5. 選択的なデータ利用: アプリ開発者は、ユーザーが特定の写真やビデオに対してのみアクセスを許可する機能を提供することで、ユーザーエクスペリエンスを向上させつつ、ユーザーデータの保護を実現することができる。これにより、データの利用が目的に応じて最適化され、不必要なデータ収集が防止される。

以下は、その具体的な実装例である。

```
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()
    )
)
```

このコードにより、アプリは部分的なメディア権限が付与された際に、ユーザーが最近選択した写真やビデオへのアクセスを取得することができる。QUERY_ARG_LATEST_SELECTION_ONLY 引数を使用することで、クエリ結果は最新の選択内容に限定され、QUERY_ARG_SQL_SORT_ORDER 引数によって結果が時系列順に並べられる。これにより、ユーザーが最近追加したメディアファイルに効率的にアクセスできるようになる。

4.6.3.11 MediaStore バージョンのロックダウン

Android 16（API Level 36）から、MediaStore#getVersion() の返す値の仕様が大きく変わった。従来、このメソッドは端末ごとに一意なバージョン文字列を返しており、その中には UUID のような端末やユーザーを特定できる情報が含まれていた。

この仕様には大きな問題があった。アプリがこのバージョン値を利用して端末やユーザーを識別できてしまい、MediaStore#getVersion() の値をサーバーなどに送信することで、ユーザーの行動追跡や端末のフィンガープリント（＝特定）が可能になっていた。

Android 16 以降、このリスクに対応するため、MediaStore#getVersion() の返却値は「アプリごとに一意」なものに変更された。新しい仕様では、返される値は「149485ba50792615」のような短い 16 進数文字列となり、端末やユーザーを特定できる情報は一切含まれない。同じ端末上でもアプリごとに異なる値が返されるため、アプリ間で値を比較しても意味がない。

このバージョン値は、MediaStore の状態が大きく変化したかどうかを検知するために使うことが想定されている。たとえば、アプリが MediaStore のデータをキャッシュしている場合、このバージョン値が変わったタイミングでキャッシュの再構築や再同期を行うといった用途だ。値の内容や形式、変化のタイミングなど、内部実装に依存した処理は避けるべきであり、あくまで「バージョン値が変わったら何か大きな変化があった」とみなして処理するのが正しい使い方となる。

```
val version = MediaStore.getVersion(this)
Log.d("MediaStoreVersion", "MediaStore#getVersion() = $version")
```

```
Target SDK 35 の場合
2025-05-26 14:42:10.272 28868-28868 MediaStoreVersion       com.example.myapplication     ␣
↪       D  MediaStore#getVersion() = 1506:f4cbc590-2645-4417-9177-99db10dcafbb

Target SDK 36 の場合
2025-05-26 14:47:22.408 29089-29089 MediaStoreVersion       com.example.myapplication     ␣
↪       D  MediaStore#getVersion() = 149485ba50792615
```

4.6.3.12 App-owned photos

Android 16（API Level 36）では、アプリが写真や動画のストレージ権限（例：READ_MEDIA_IMAGES）をリクエストし、ユーザーが「選択したメディアのみ」を許可した場合、アプリが MediaStore 経由で保存した写真や動画（App-owned photos）が権限許可ダイアログで自動的に選択状態（プリセレクト）になる。ユーザーはこのプリセレクトを解除でき、解除された場合アプリはその写真や動画へのアクセス権を失う。プリセレクトは「権限許可ダイアログ」でのみ発生し、通常の Photo Picker API 利用時には確認できない。Photo Picker のみを使うアプリではこの挙動自体が発生しない。

開発者はこの仕様を理解し、権限リクエスト時に自アプリの写真がプリセレクトされること、ユーザーがその選択を解除できることを設計や UI、説明文で明示する必要がある。ユーザーがプリセレクトを解除した場合、その写真へのアクセスができなくなるため、画像や動画へのアクセス時には必ず権限の有無を確認し、アクセスできない場合は設定アプリへの誘導や再選択の案内など、適切な例外処理やガイダンスを実装する。Google は Photo Picker の利用を推奨しており、一般的なアプリではストレージ権限を宣言しない設計が推奨される。ギャラリーアプリなど特殊な用途でストレージ権限が必要な場合のみ、この仕様を考慮して実装する。

実際の検証では、MediaStore 経由でアプリ所有の写真を保存し、ストレージ権限をリクエストすることで、権限許可ダイアログにおいて自アプリの写真がプリセレクトされていることを確認できる。ユーザーがこのチェックを外して権限

を許可した場合、その写真へのアクセス権が失われることもアプリ側で確認できる。再度ストレージ権限をリクエストしてもダイアログは表示されず、許可状態の変更は Android の設定アプリからのみ可能となる。

以下はこの仕様の検証に使った Compose サンプルコード例。

```
@Composable
fun AppOwnedPhotoPermissionTestScreen(modifier: Modifier = Modifier) {
    val context = LocalContext.current
    var savedUri by remember { mutableStateOf<android.net.Uri?>(null) }
    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) {
            "権限が許可された（プリセレクト挙動を確認）"
        } else {
            "権限が拒否された"
        }
        if (isGranted && savedUri != null) {
            try {
                context.contentResolver.openInputStream(savedUri!!)?.use { input ->
                    bitmap = android.graphics.BitmapFactory.decodeStream(input)
                }
                message += "\n保存した写真を表示"
            } catch (e: Exception) {
                // ここに適切な例外処理とユーザーへのガイダンスを実装すること
                // 例：設定アプリで権限を確認・変更するよう促すダイアログ表示など
                message += "\n写真へのアクセスに失敗しました。設定から権限を確認してください。"
            }
        }
    }

    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")
            }
```

（次のページに続く）

(前のページからの続き)

```
            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 = "アプリ所有の写真を保存"
            } else {
                message = "写真の保存に失敗"
            }
        }) {
            Text("アプリ所有の写真を保存")
        }

        Spacer(Modifier.height(16.dp))

        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("ストレージ権限をリクエスト")
        }

        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("保存した写真の URI: $savedUri", style = MaterialTheme.typography.bodySmall)
        }
    }
}
```

4.7 Intent を利用する

Intent は、Android におけるコンポーネント間通信の基本的な仕組みであり、アプリ内外の Activity や Service、Broadcast Receiver などを起動する際に利用される。Intent には明示的 Intent と暗黙的 Intent が存在し、用途やセキュリティ要件に応じて使い分ける必要がある。

明示的 Intent は、起動対象のコンポーネントクラスを明確に指定して呼び出すもので、主にアプリ内部で利用される。一方、暗黙的 Intent は、アクションやデータタイプなどを指定し、条件に合致するコンポーネントがシステムによって選択される。これにより、他アプリとの連携やシステム標準機能の呼び出しが可能となる。

Intent の利用時には、不要な情報の漏洩や権限の過剰付与、exported 属性や Intent Filter の設定ミスなど、セキュリティ上のリスクに留意する必要がある。特に暗黙的 Intent を利用する場合、想定外のアプリが受信する可能性があるため、送信する情報や受信時のバリデーションを徹底することが求められる。

暗黙的 Intent の中でも、android.intent.category.BROWSABLE カテゴリを付与したものは「Browsable Intent」と呼ばれる。Browsable Intent は、主に Web ブラウザや他アプリからのディープリンク（例: https スキームの URL）を受けて Activity を起動する用途で利用される。Manifest の intent-filter に BROWSABLE カテゴリを指定することで、外部からの呼び出しを許可することができる。

Browsable Intent を受け付ける Activity は、外部からの入力を受けることになるため、Intent で渡されるデータやパラメータの検証を必ず行い、不正なリクエストや情報漏洩を防ぐ設計とする。不要な Activity に BROWSABLE カテゴリを付与しない、パラメータのバリデーションやアクセス制御を適切に実装するなど、セキュリティ対策を徹底すること。

Intent 全般の基本と注意点を押さえた上で、Browsable Intent の設計・実装にあたっては、外部連携ポイントとしてのリスクを十分に認識し、適切な対策を講じることが重要となる。

4.7.1 サンプルコード

以下に、Browsable Intent を利用したアプリのサンプルコードを示す。Starter.html を Web サーバーにインストールして実行する。

ポイント：

1. (Web ページ側) 対応する URI スキーマを使ったリンクのパラメータにセンシティブな情報を含めない
2. URL のパラメータを利用する前に値の安全性を確認する

```
Starter.html
<html>
    <body>
<!-- ★ポイント 1★ (Web ページ側) 対応する URI スキーマを使ったリンクのパラメータにセンシティブな情報を含めない -->
<!-- URL パラメータとして渡す文字列は、UTF-8 で、かつ URI エンコードしておくこと -->
        <a href="secure://jssec?user=user_id"> Login </a>
    </body>
</html>
```

```
AndroidManifest.xml
<?xml version="1.0" encoding="utf-8"?>
```

(次のページに続く)

(前のページからの続き)

```
<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" />
        <!-- 暗黙的 intent を受け付ける -->
        <category android:name="android.intent.category.DEFAULT" />
        <!-- Browsable intent を受け付ける -->
        <category android:name="android.intent.category.BROWSABLE" />
        <!-- URI 'secure://jssec' を受け付ける -->
        <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");
 * 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
```

(次のページに続く)

（前のページからの続き）

```
 * 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) {
            // URL パラメータで渡されたユーザー ID を取得する
            // ★ポイント 2★ URL のパラメータを利用する前に値の安全性を確認する
            // サンプルにつき割愛。「3.2 入力データの安全性を確認する」を参照。
            String userID = "User ID = " +  uri.getQueryParameter("user");
            TextView tv = (TextView)findViewById(R.id.text_userid);
            tv.setText(userID);
        }
    }
}
```

4.7.2 ルールブック

Browsable Intent を利用する場合には以下のルールを守ること。

1. （*Web* ページ側）対応するリンクのパラメータにセンシティブな情報を含めない（必須）
2. *URL* のパラメータを利用する前に値の安全性を確認する（必須）
3. 可能であれば *Android App Links* を利用する （推奨）

4.7.2.1 （Web ページ側）対応するリンクのパラメータにセンシティブな情報を含めない（必須）

ブラウザ上でリンクをタップした際、data（Intent#getData にて取得) に URL の値が入った Intent が発行され、システムにより該当する Intent Filter を持つアプリが起動する。

この時、同じ URI スキームを受け付けるよう Intent Filter が設定されたアプリが複数存在する場合は、通常の暗黙的 Intent による起動と同様にアプリ選択のダイアログが表示され、ユーザーの選択したアプリが起動することになる。仮に、アプリ選択画面の選択肢としてマルウェアが存在していた場合は、ユーザーが誤ってマルウェアを起動させてしまう危険性があり、パラメータがマルウェアに渡ることになる。

このように Web ページのリンク URL に含めたパラメータはすべてマルウェアに渡る可能性があるので、一般の Web ページのリンクを作るときと同様に、URL のパラメータに直接センシティブな情報を含めることは避けなければならない。

URL にユーザー ID とパスワードが入っている例 insecure://sample/login?userID=12345&password=abcdef

また、URL のパラメータがユーザー ID などセンシティブでない情報のみの場合でも、アプリ起動時のパスワード入力をアプリ側でさせるような仕様では、ユーザーが気付かずにマルウェアを起動してしまい、マルウェアに対してパスワードを入力してしまう危険性もある。そのため、一連のログイン処理自体はアプリ側で完結するような仕様を検討すべきである。Browsable Intent によるアプリ起動はあくまで暗黙的 Intent によるアプリ起動であり、意図したアプリが起動される保証がないことを念頭に置いたアプリ・サービス設計を心がける必要がある。

4.7.2.2 URL のパラメータを利用する前に値の安全性を確認する（必須）

URI スキーマに合わせたリンクは、アプリ開発者に限らず誰でも作成可能なので、アプリに渡された URL のパラメータが正規の Web ページから送られてくるとは限らない。また、渡された URL のパラメータが正規の Web ページから送られてきたかどうかを調べる方法もない。

そのため、渡された URL のパラメータを利用する前に、パラメータに想定しない値が入っていないかなど、値の安全性を確認する必要がある。

4.7.2.3 可能であれば Android App Links を利用する （推奨）

Android には App Links という仕組みがあり、Web サイトの URL とアプリを紐付けることが可能である。URL とアプリを紐付けることによって、同一のリンクを指定しているアプリが端末内に 2 つ以上存在する場合であっても、URL に紐付く正規のアプリを特定することができる。そのため、端末に同一のリンクを指定したマルウェアアプリがインストールされていたとしても、リンクからマルウェアアプリが起動することはなく、ユーザの選択なしで正規のアプリが起動する。これは Android 6.0（API Level 23）以降の端末で利用可能である。

App Links を利用するためにはアプリと Web サイトの両方の所有権を証明する必要がある。そのため開発者は、アプリのパッケージ名や証明書のフィンガープリントなどを記載した json ファイルを用意し、自社の Web サイト上で公開する必要がある。そして Browsable Intent の Intent Filter に自社の Web サイトと同様のリンクを指定し、android:autoVerify=”true” を宣言することで、リンクが検証され利用可能となる。

```
https://www.jssec.org/.well-known/assetlinks.json
[{
    "relation": ["delegate_permission/common.handle_all_urls"],
    "target": {
        "namespace": "android_app",
```

(次のページに続く)

(前のページからの続き)

```
        "package_name": "アプリのパッケージ名",
        "sha256_cert_fingerprints":
        ["証明書のフィンガープリント"]
    }
}]
```

```
AndroidManifest.xml
    <activity
        android:name=".BrowsableIntentActivity"
        android:exported="true" >
      <!-- android:autoVerify="true"を宣言する -->
      <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' を受け付ける -->
        <data android:scheme="https" android:host="jssec.org" />
      </intent-filter>
    </activity>
```

4.7.3 アドバンスト

4.7.3.1 コンポーネント公開制御と Intent 送信の制限

Android 12（API Level 31）以降、intent-filter を定義したコンポーネント（Activity、Service、Broadcast Receiver）は、android:exported 属性を明示的に設定しないとビルドできない。exported 属性を省略すると、マニフェスト編集時に警告が表示され、ビルド時にはエラーになる。intent-filter の有無によって暗黙的にコンポーネントの公開／非公開が決まっていた従来の挙動は廃止され、設計ミスによるセキュリティリスクを防ぐための仕様となった。exported=”true” なら外部から呼び出し可能、exported=”false” なら外部から呼び出せない。どちらを指定するかは設計意図に合わせて決める。対象は Activity、Service、Broadcast Receiver すべて。intent-filter を使う場合は必ず exported 属性を明示すること。

```
<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>
```

Android 14 以降では、exported=”false” の内部コンポーネントに暗黙的 Intent を送信した場合、例外が発生するようになった。たとえば、非公開 Activity に対して暗黙的 Intent を送ると、Android 13 までは起動できていたが、Android 14 以降は ActivityNotFoundException となる。これは、Intent Filter の定義が他のアプリとバッティングしていた場合、意図しない Activity を呼び出せてしまうリスクを防ぐためである。非公開コンポーネントには setPackage や setClassName で明示的 Intent を使うこと。exported 属性・intent-filter・Intent 種別（明示的/暗黙的）の設計は一体で考える必要がある。

```
Intent intent = new Intent("com.example.myapplication.MyAction");
intent.setPackage(getApplicationContext().getPackageName());
startActivity(intent);
```

4.7.3.2 PendingIntent オブジェクトの可変性

PendingIntent は、アプリが自分の権限で他のアプリやシステムに「後で実行してほしい Intent」を委譲するための仕組みである。生成時に Intent と各種フラグを指定するが、特にセキュリティ上重要なのが「可変性（mutability）」の指定。可変（mutable）な PendingIntent は、委譲先アプリやシステムが Intent の一部を変更できる。fillIn() などで未設定フィールドを上書きできるため、悪意あるアプリに本来想定していない動作をさせられるリスクが生じる。

Android 12 以降をターゲットとするアプリでは、PendingIntent オブジェクトの可変性の指定が必須。可変の場合は PendingIntent.FLAG_MUTABLE、不変の場合は PendingIntent.FLAG_IMMUTABLE フラグにより行う。いずれのフラグも指定せずに PendingIntent オブジェクトを作成しようとすると IllegalArgumentException が発生する。セキュリティ強化の観点から FLAG_IMMUTABLE の指定が推奨されるが、ダイレクト返信アクションなど一部の機能では FLAG_MUTABLE 指定が必要となる。

```
PendingIntent pendingIntent = PendingIntent.getActivity(getApplicationContext(), REQUEST_
↪CODE, intent, PendingIntent.FLAG_IMMUTABLE);
// ダイレクト返信等、mutable が必要な場合
PendingIntent pendingIntent = PendingIntent.getActivity(getApplicationContext(), REQUEST_
↪CODE, intent, PendingIntent.FLAG_MUTABLE);
```

4.7.3.3 Intent Filter の厳格化

Android 13 (API Level 33) 以降、明示的 Intent で Activity を起動する際、受信する Activity の intent-filter のいずれにも一致しない場合は、その Intent がブロックされる。ただし、intent-filter が定義されていないコンポーネント、同一アプリケーション内から送信された Intent、システムや root から送信された Intent にはこの制約は適用されない。

具体例として、以下のように intent-filter を定義した公開 Activity を作成した。マッチしない Intent のブロックを確認するために、adb コマンドで異なる action の明示的 Intent を使用し公開 Activity を起動させる。ここで、intent-filter に定義されていない not_match_intent_filter を使用する場合 Activity が起動できないことを確認できた。

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">
        <!--公開 Activity-->
```

(次のページに続く)

(前のページからの続き)

```
        <!--intent-filter が定義されている-->
        <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>

</manifest>
```

adb コマンドによる Activity 起動の確認

```
>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.
```

(次のページに続く)

(前のページからの続き)

```
>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 }
```

また、Android 15 以降、Intent をより安全かつ堅牢にするために、Intent Filter への完全一致やアクション必須化などの新しいセキュリティ対策が導入された。ターゲットの Intent Filter に一致しない Intent は受信できなくなり、アクションのない Intent も Intent Filter と一致しなくなる。PendingIntent にも同様の制約が適用される。Intent Filter の action、data、category すべてに一致させる設計とし、Intent には必ずアクションを設定する。

```
<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)
```

公開 Activity の intent-filter 設計は必要最小限とし、不要な受信口を作らない設計とする。StrictMode の detectUnsafeIntentLaunch() を有効にして、マッチング違反やアクション未指定 Intent 送信がないか検証する。

```
StrictMode.setVmPolicy(
    new StrictMode.VmPolicy.Builder()
        .detectUnsafeIntentLaunch()
        .penaltyLog()
        .build()
);
```

4.7.3.4 Intent Filter 機能の拡張

Android 15 で UriRelativeFilter と UriRelativeFilterGroup が導入され、従来のスキーム・ホスト・パスに加え、クエリパラメータやフラグメントを含む詳細なマッチングが可能になった。認証フローやキャンペーン等、特定の用途で特定のクエリパラメータを持ったリクエストのみを受け付けるといった柔軟な受信制御が実現できる。フィルタ条件は必要最小限とし、不要なリクエストを受け付けない設計とする。adb 等で受信挙動を検証し、意図しない URI で起動されないかテストしておく。

```
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
```

フィルタにマッチする URI の場合：

```
$ adb shell am start -W -a android.intent.action.VIEW -d "https://example.com/auth?
↪token=securetoken"
```

フィルタにマッチしない URI の場合：

```
$ adb shell am start -W -a android.intent.action.VIEW -d "https://example.com/auth?
↪token=invalidtoken"
```

4.7.3.5 PendingIntent とパッケージ停止状態の仕様変更

パッケージの FLAG_STOPPED 状態（アプリアイコンを長押しして「Force stop」を選択することでユーザーがこの状態に移行する）は、ユーザーが直接アプリを起動するか、共有シートやウィジェットを介してアプリと間接的にやり取りするなどの方法でこの状態からアプリを明示的に解除するまで、アプリをこの状態に保つことを目的としている。

これまでのバージョンの Android では、アプリが停止状態に入っても、保留中の PendingIntent は引き続き有効であり、システムはこれらの PendingIntent をキャンセルしないままであった。これにより、停止状態にあるアプリが意図せず再びアクティブになる可能性があった。

Android 15 では、FLAG_STOPPED 状態の本来の仕様に合わせてシステムの挙動が変更となり、アプリは、ユーザーの直接的または間接的なアクションによってのみ、停止状態から解除される仕組みとなっている。

この変更により、既存のアプリは以下の影響を受ける可能性がある：

1. アプリが停止状態に入った場合、すべての PendingIntent がキャンセルされるため、重要な通知やアクティビティが失われる可能性がある。
2. ウィジェットを利用しているアプリは、ウィジェットがグレーアウトされ、ユーザーがそれらとやり取りできなくなる。

既存のアプリは、これらの影響に対応するために、アプリが再度起動されたときに PendingIntent を再登録するロジックを追加する必要がある。

```
override fun onReceive(context: Context, intent: Intent) {
    if (Intent.ACTION_BOOT_COMPLETED == intent.action) {
        // Handle the app restart after boot completed
        reinitializePendingIntents(context)
    }
}

private fun reinitializePendingIntents(context: Context) {
```

(次のページに続く)

(前のページからの続き)

```
    // 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)
}
```

FLAG_STOPPED 状態の確認は ApplicationStartInfo.wasForceStopped() メソッドにより行える。

```
if (ApplicationStartInfo.wasForceStopped()) {
    // Handle the case where the app was force stopped
    reinitializePendingIntents()
}
```

4.7.3.6 Safer Intents

Safer Intents は、Android 16 で導入された Intent のセキュリティ強化機能である。manifest の android:intentMatchingFlags="enforceIntentFilter" を有効にすると、明示的 Intent（コンポーネント名指定）であっても Intent Filter にマッチしない Intent は受信できなくなる。従来は action 未指定でも明示的 Intent なら受信可能だったが、この仕様により意図しない Intent の受信や情報漏洩リスクを抑制できる。本機能は複数アプリ間の通信にのみ適用され、同一アプリ内の Intent 送信には影響しない。

```
<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")
// action あり
intent.action = "com.example.MY_CUSTOM_ACTION"
sendBroadcast(intent)
// action なし
```

(次のページに続く)

(前のページからの続き)

```
val intentNoAction = Intent()
intentNoAction.component = ComponentName("com.example.myapplication", "com.example.
↪myapplication.MyBroadcastReceiver")
sendBroadcast(intentNoAction)
```

このコードを実行すると、action ありの明示的 Intent のみ受信され、action なし Intent は受信されない。Safer Intents の有効化により、意図しない Intent 受信を確実にブロックできる。

この仕組みは、今後の Android プラットフォームにおける Intent 通信のセキュリティ強化のロードマップの一環として位置づけられている。将来的にはデフォルトで有効化される可能性が高く、現時点でも Google は新規・既存問わずアプリ開発時に積極的な導入を推奨している。

4.7.3.7 Intent リダイレクト攻撃対策

Activity の exported 属性を”false” と明示することで、他アプリからの直接的な起動を防ぐことができる。しかし、Android 15（API Level 35）までの環境では、他アプリから Intent の extra 経由でサブレベル Intent として非公開 Activity を渡された場合、startActivity によって意図せず起動できてしまうという問題が存在していた。これは、exported 属性による非公開設定が「直接起動」には有効であっても、Intent の extra 等を介したリダイレクト経由の起動まではシステム側で十分に防御されていなかったためである。

この問題に対し、Android 16（API Level 36）ではシステムレベルでの防御が導入された。他アプリから受け取った Intent の extra に Intent が含まれている場合（サブレベル Intent）、それをそのまま startActivity 等で起動しようとすると、デフォルトでセキュリティ例外が発生し、起動がブロックされるようになった。これにより、exported=”false” とした Activity や本来外部に公開すべきでないコンポーネントが、extra 経由で意図せず起動されるリスクが原則として封じられることとなった。やむを得ずこの保護を解除したい場合は、removeLaunchSecurityProtection() を明示的に呼び出すことでのみ起動が可能となるが、この API は Android 16 以降でのみ利用できる。

実際の検証では、senderapp から receiverapp の非公開 Activity（exported=”false”）を指す Intent を extra（”sub_intent”）に格納し、receiverapp の MainActivity でこれをそのまま startActivity する、あるいは removeLaunchSecurityProtection() を呼んでから startActivity する、という 2 パターンを用意した。検証コード例を以下に示す。

senderapp 側（攻撃シナリオの送信側）

```
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 側（受信・起動側）

```
val iSublevel = intent.getParcelableExtra("sub_intent", Intent::class.java)
try {
```

(次のページに続く)

(前のページからの続き)

```
    // Android 16 ではここでセキュリティ例外が発生し、PrivateActivity は起動できない
    iSublevel?.let { startActivity(it) }
} catch (e: Exception) {
    // 例外内容をログや UI に表示
}

try {
    // Android 16 以降のみ有効。removeLaunchSecurityProtection() を呼ぶと保護を解除できる
    iSublevel?.removeLaunchSecurityProtection()
    iSublevel?.let { startActivity(it) }
} catch (e: Exception) {
    // 例外内容をログや UI に表示
}
```

この検証の結果、Android 15 ではそのまま startActivity を呼ぶことで PrivateActivity が起動できてしまう一方、Android 16 では removeLaunchSecurityProtection() を呼ばない限りセキュリティ例外でブロックされることが確認できた。また、Android 15 環境で removeLaunchSecurityProtection() を呼ぶと NoSuchMethodError となりアプリがクラッシュするため、API レベルによる分岐実装が必須となる。

表 4.7.1　Android 15/16 における Intent リダイレクト攻撃対策の挙動

手順	**Android 15（API Level 35）**	**Android 16（API Level 36）**
そのまま startActivity	起動できてしまう	セキュリティ例外でブロックされる
removeLaunchSecurityProtection	NoSuchMethodError でクラッシュ（API 未実装）	保護を解除し、PrivateActivity が起動

開発者は、Android 16 以降ではこのセキュリティ強化を前提に設計・実装を行うべきであり、特に他アプリから受け取った Intent の extra に含まれる Intent をそのまま起動しない、送り先や内容の検証・サニタイズを徹底することが求められる。非公開 Activity については exported=”false” の明示だけでなく、extra 経由でのリダイレクトにも十分注意し、不要な外部公開や意図しない起動が発生しないよう設計上配慮する必要がある。removeLaunchSecurityProtection() の使用は最小限にとどめ、やむを得ず利用する場合は十分なリスク評価とテストを行うことが望ましい。

4.7.3.8 暗黙的な URI 権限付与の制限

Android 18 以降では、他のアプリと URI を共有する際の暗黙的な URI 権限付与（Implicit URI Grants）が制限される予定である。この変更は、アプリ間連携における意図しないファイルアクセスを防ぎ、セキュリティを強化することを目的としている。

従来、アプリ間でファイル（URI）を共有する場合、例えば Intent.ACTION_SEND などのインテントに URI を含めて送信すると、システムが自動的（暗黙的）に受信アプリに対して読み取り（READ）や書き込み（WRITE）の権限を付与していた。しかし、Android 18 以降の制限下ではこの暗黙的な権限付与が行われなくなる。そのため、これまで通り権限付与のフラグを省略したまま送信すると、受信側のアプリがファイルにアクセスできず、SecurityException などのエラーが発生したり機能が正しく動作しなくなる。Android 17 の環境で暗黙的な URI 権限付与を行うと、Logcat に Implicit URI grant for android.intent.action.SEND action will be discontinued from Android 18 onwards. Please set the grant explicitly in the app. のようなシステムエラーログが出力され、デベロッパーに対応が促される。

対応方法として、送信側のアプリで明示的にアクセス権限を付与するフラグ（ Intent.FLAG_GRANT_READ_URI_PERMISSION または Intent.FLAG_GRANT_WRITE_URI_PERMISSION ）を設定する必要がある。また、より確実に URI の権限を受信アプリに委譲するため、 Intent.EXTRA_STREAM だけでなく clipData にも URI をセットすることが推奨される。

特に以下のような機能を持つアプリは、明示的な権限付与が行われているか確認し、対応を行うことが強く推奨される。

- 写真や PDF などのファイル共有機能
- 他アプリへのファイル送信・メール添付作成
- SNS 等への共有機能
- カメラ起動（ ACTION_IMAGE_CAPTURE による画像保存先 URI の受け渡し）

以下に、非推奨な実装と、明示的に権限を付与する Android 18 対応の実装例を示す。

```
// NG 例：従来の非推奨な実装（暗黙的な権限付与に依存）
fun shareFileLegacy(context: Context, fileUri: Uri) {
    val intent = Intent(Intent.ACTION_SEND).apply {
        putExtra(Intent.EXTRA_STREAM, fileUri)
        type = "image/jpeg"
        // 従来はシステムが自動で権限を付与してくれていたためフラグを省略しがちだった
    }
    val chooser = Intent.createChooser(intent, "フラグなしで共有")
    context.startActivity(chooser)
}

// OK 例：Android 18 対応の実装（明示的な権限付与と ClipData の利用）
fun shareFileSafely(context: Context, fileUri: Uri) {
    val intent = Intent(Intent.ACTION_SEND).apply {
        putExtra(Intent.EXTRA_STREAM, fileUri)
        type = "image/jpeg"

        // ① Android 18 以降の制限を回避するため、明示的に READ 権限フラグを付与する
        addFlags(Intent.FLAG_GRANT_READ_URI_PERMISSION)

        // ② さらに確実に権限を委譲するため、ClipData にも URI をセットする（推奨実装）
        clipData = ClipData.newRawUri("", fileUri)
    }
    val chooser = Intent.createChooser(intent, "権限を付与して共有")
    context.startActivity(chooser)
}
```

なお、カメラアプリを起動して写真を撮影し、自アプリが用意した URI に保存させる場合（ ACTION_IMAGE_CAPTURE ）は、カメラアプリに対してファイルの書き込み権限を与える必要があるため、 intent.addFlags(Intent.FLAG_GRANT_WRITE_URI_PERMISSION) を明示的に指定して起動する。

4.7.3.9 プロファイル間のループバック制限

Android 17 以降では、個人プロファイルと仕事用プロファイルなど、プロファイル間のループバックトラフィック（localhost / 127.0.0.1）がデフォルトで制限される。

プロファイルとは、Android 端末の中でユーザーや用途ごとにアプリやデータを分離して管理するための仕組みであり、1 台のスマートフォンの中に複数の独立した利用環境を持たせることができる機能である。代表的なものとしては、個人が自由に使う「個人プロファイル」と、企業が管理する「仕事用プロファイル（Work Profile）」があり、それぞれのプロファイルは別々の領域として扱われる。同じアプリであってもプロファイルごとに別インストールとして動作し、保存されるデータや設定、アクセスできる情報も分離されるため、情報漏えいや不正アクセスのリスクを低減できる。

セキュリティ強化の背景

従来のバージョンでは、同一端末内であれば別プロファイルのアプリ同士でも、ローカルホスト経由で通信できてしまうケースがあり、これが意図しないデータ共有や情報漏えいの経路になる可能性があった。特に、片方のプロファイルで動作するアプリがローカルサーバとして待ち受け、もう一方のプロファイルのアプリがそこに接続することで、本来分離されているはずのデータ領域を横断できてしまう点が問題とされていた。

Android 17 におけるプロファイル間ループバック通信の制限により、同じ端末内でも別ユーザー扱いとして厳密に分離される。企業のワークプロファイルに保存された業務データが個人プロファイル側のアプリに流出するリスクや、マルウェアが別プロファイル経由で情報を取得するような攻撃手法が大幅に制限される。

開発者への影響と推奨される代替手段

これまでローカル HTTP サーバや localhost ベースの連携でプロファイル間通信を実現していた場合、それらの実装が動作しなくなる可能性があるため注意が必要である（同じプロファイル内でのループバック通信は影響を受けない）。

プロファイルをまたぐ連携が必要な場合の代替手段としては、Android が標準で提供する CrossProfileApps API や、権限制御された Intent、ContentProvider などを利用し、明示的かつ安全にプロファイル間連携を行う設計への移行が求められる。この変更は、利便性よりもセキュリティと分離性を優先する方向への重要な仕様変更であり、企業利用や高セキュリティ環境を前提としたアプリ開発において特に留意すべきポイントとなる。

4.8 LogCat にログ出力する

Android は LogCat と呼ばれるシステムログ機構があり、システムのログ情報だけでなくアプリのログ情報も LogCat に出力される。LogCat のログ情報は同じ端末内の他のアプリからも読み取り可能[*27] であるため、センシティブな情報を LogCat にログ出力してしまうアプリには情報漏洩の脆弱性があるとされる。LogCat にはセンシティブな情報をログ出力すべきではない。

セキュリティ観点ではリリース版アプリでは一切ログ出力しないことが望ましい。しかし様々な理由によりリリース版アプリでもログ出力するケースがある。ここではリリース版アプリにおいてもログ出力しつつ、センシティブな情報はログ出力しない方法を紹介する。また「4.8.3.1. リリース版アプリにおけるログ出力の 2 つの考え方」も参照すること。

[*27] LogCat に出力されたログ情報は、READ_LOGS Permission を利用宣言したアプリであれば読み取り可能である。ただし Android 4.1 以降では LogCat に出力された他のアプリのログ情報は読み取り不可となった。また、スマートフォンユーザーであれば、ADB 経由で LogCat のログ情報を参照することも可能である。

4.8.1 サンプルコード

ここでは ProGuard を利用してリリース版アプリでの LogCat へのログ出力を制御する方法を紹介する。ProGuard は使用されていないメソッド等、実質的に不要なコードを自動削除する最適化ツールの一つである。

Android の android.util.Log クラスには 5 種類のログ出力メソッド Log.e()、Log.w()、Log.i()、Log.d()、Log.v() がある。ログ情報は、リリース版アプリで出力することを意図したログ情報（以下、運用ログ情報と呼ぶ）と、リリース版アプリで出力してはならない（たとえばデバッグ用の）ログ情報（以下、開発ログ情報と呼ぶ）を区別するべきである。運用ログ出力のためには Log.e()/w()/i() を使用し、開発ログ出力のためには Log.d()/v() を使用するとよい。5 種類のログ出力メソッドの使い分けの詳細については「4.8.3.2. ログレベルとログ出力メソッドの選択基準」を参照すること。また、「4.8.3.3. *DEBUG* ログと *VERBOSE* ログは自動的に削除されるわけではない」も参照すること。

次ページ以降で、Log.d()/v() で出力する開発ログ情報を開発版アプリではログ出力し、リリース版アプリではログ出力しないサンプルコードを紹介する。このサンプルコードでは Log.d()/v() 呼び出しコードを自動削除するために、ProGuard を使用している。

ポイント：

1. センシティブな情報は Log.e()/w()/i()、System.out/err で出力しない
2. センシティブな情報をログ出力する場合は Log.d()/v() で出力する
3. Log.d()/v() の呼び出しでは戻り値を使用しない (代入や比較)
4. リリースビルドでは Log.d()/v() の呼び出しが自動削除される仕組みを導入する
5. リリース版アプリの APK ファイルはリリースビルドで作成する

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
 *
 * 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;
```

(次のページに続く)

(前のページからの続き)

```
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);

        // ★ポイント 1★ センシティブな情報は Log.e()/w()/i()、System.out/err で出力しない
        Log.e(LOG_TAG, "センシティブではない情報 (ERROR)");
        Log.w(LOG_TAG, "センシティブではない情報 (WARN)");
        Log.i(LOG_TAG, "センシティブではない情報 (INFO)");

        // ★ポイント 2★ センシティブな情報をログ出力する場合は Log.d()/v() で出力する
        // ★ポイント 3★ Log.d()/v() の呼び出しでは戻り値を使用しない (代入や比較)
        Log.d(LOG_TAG, "センシティブな情報 (DEBUG)");
        Log.v(LOG_TAG, "センシティブな情報 (VERBOSE)");
    }
}
```

```
proguard-project.txt
# クラス名、メソッド名等の変更を防ぐ
-dontobfuscate

# ★ポイント 4★ リリースビルドでは Log.d()/v() の呼び出しが自動削除される仕組みを導入する
-assumenosideeffects class android.util.Log {
    public static int d(...);
    public static int v(...);
}
```

★ポイント 5 ★ リリース版アプリの APK ファイルはリリースビルドで作成する

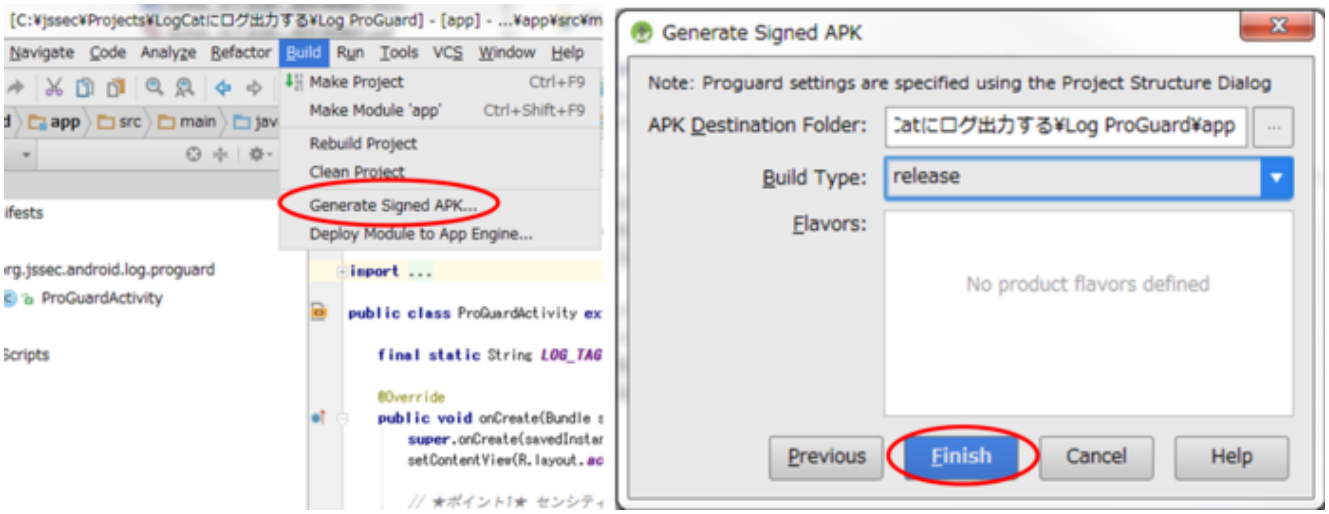

図 4.8.1　リリース版アプリを作成する方法（Export する）

開発版アプリ（デバッグビルド）とリリース版アプリ（リリースビルド）の LogCat 出力の違いを 図 4.8.2 に示す。

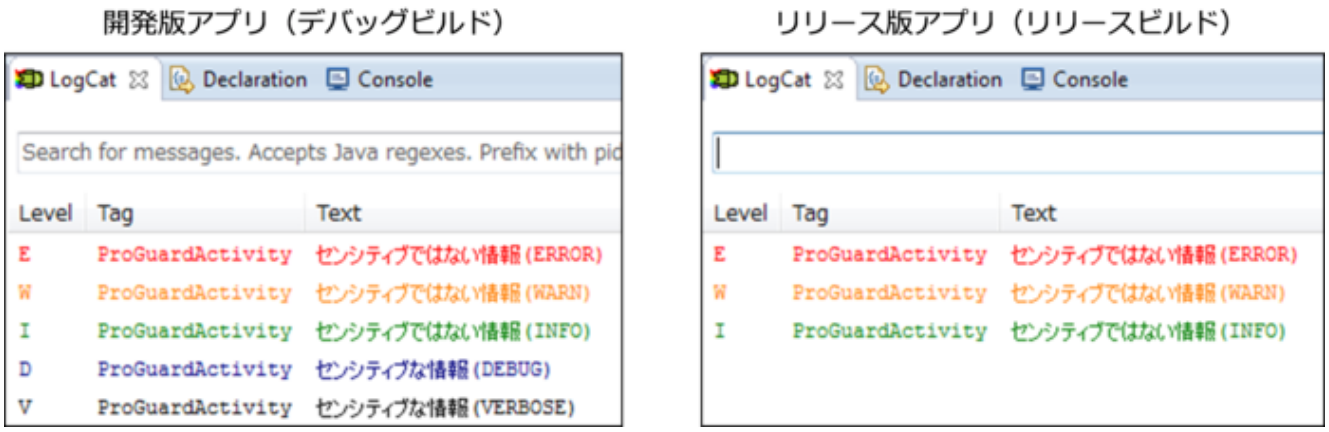

図 4.8.2　Log メソッドの開発版アプリとリリース版アプリの LogCat 出力の違い

4.8.2 ルールブック

LogCat にログを出力する際は、以下のルールを守ること。

1. 運用ログ情報にセンシティブな情報を含めない（必須）
2. 開発ログ情報を出力するコードをリリースビルド時に自動削除する仕組みを導入する （推奨）
3. *Throwable* オブジェクトをログ出力するときは *Log.d()/v()* メソッドを使う （推奨）
4. ログ出力には *android.util.Log* クラスのメソッドのみ使用する （推奨）

4.8.2.1 運用ログ情報にセンシティブな情報を含めない（必須）

LogCat に出力したログは他のアプリから読むことができるので、リリース版アプリがユーザーのログイン情報などのセンシティブな情報をログ出力することがあってはならない。開発中にセンシティブな情報をログ出力するコードを書かないようにするか、あるいは、リリース前にそのようなコードを全て削除することが必要である。

このルールを順守するためには、運用ログ情報にセンシティブな情報を含めないこと。さらに、センシティブな情報を出

力するコードをリリースビルド時に削除する仕組みを導入することを強く推奨する。「4.8.2.2. 開発ログ情報を出力するコードをリリースビルド時に自動削除する仕組みを導入する （推奨）」を参照すること。

4.8.2.2 開発ログ情報を出力するコードをリリースビルド時に自動削除する仕組みを導入する （推奨）

アプリ開発中は、複雑なロジックの処理過程の中間的な演算結果、プログラム内部の状態情報、通信プロトコルの通信データ構造など、処理内容の確認やデバッグ用でセンシティブな情報をログ出力させたいことがある。アプリ開発時にセンシティブな情報をデバッグログとして出力するのは構わないが、この場合は、「4.8.2.1. 運用ログ情報にセンシティブな情報を含めない（必須）」で述べたように、リリース前に必ず該当するログ出力コードを削除すること。

リリースビルド時に開発ログ情報を出力するコードを確実に削除するために、何らかのツールを用いてコード削除を自動化する仕組みを導入すべきである。そのためのツールに「4.8.1. サンプルコード」で紹介した ProGuard がある。以下では、ProGuard を使ったコード削除の仕組みを導入する際の注意を説明する。ここでは、「4.8.3.2. ログレベルとログ出力メソッドの選択基準」に準拠し、開発ログ情報を Log.d()/v() のいずれかのみで出力しているアプリに対して仕組みを適用することを想定している。

ProGuard は使用されていないメソッド等、実質的に不要なコードを自動削除する。Log.d()/v() を-assumenosideeffects オプションの引数に指定することにより、Log.d()、Log.v() の呼び出しが実質的に不要なコードとみなされ、自動削除される。

Log.d()/v() を-assumenosideeffects と指定することで、自動削除の対象にする

```
-assumenosideeffects class android.util.Log {
    public static int d(...);
    public static int v(...);
}
```

この自動削除の仕組みを利用する場合は、Log.v(), Log.d() の戻り値を使用してしまうと Log.v()/d() のコードが削除されない点に注意が必要である。よって、Log.v(), Log.d() の戻り値を使用してはならない。たとえば、次の実験コードにおいては、Log.v() が削除されない。

削除指定した Log.v() が削除されない実験コード

```
int i = android.util.Log.v("tag", "message");
System.out.println(String.format("Log.v() が %d を返した。", i));  // 実験のため Log.v() の戻り値を使用。
```

また、上記 ProGuard 設定により、Log.d() 及び Log.v() が自動削除されることを前提としたソースコードがあったとする。もしそのソースコードを ProGuard 設定がされていない他のプロジェクトで再利用してしまうと、Log.d() 及び Log.v() が削除されないため、センシティブな情報が漏洩してしまう危険性がある。ソースコードを再利用する際は、ProGuard 設定を含めたプロジェクト環境の整合性を確保すること。

4.8.2.3 Throwable オブジェクトをログ出力するときは Log.d()/v() メソッドを使う （推奨）

「4.8.1. サンプルコード」および「4.8.3.2. ログレベルとログ出力メソッドの選択基準」に示した通り、Log.e()/w()/i() ではセンシティブな情報をログ出力してはならない。一方で、開発者がプログラムの異常を詳細にログ出力するために、例外発生時に Log.e(..., Throwable tr)/w(..., Throwable tr)/i(..., Throwable tr) でスタックトレースを LogCat にログ出力しているケースがみられる。しかしながら、スタックトレースはプログラムの内部構造を詳細に出力してしまうので、アプリケーションによってはセンシティブな情報が含まれてしまう場合がある。例えば、SQLiteException をそのまま出力して

しまうと、どのような SQL ステートメントが発行されたかが明らかになるので、SQL インジェクション攻撃の手がかりを与えてしまうことがある。よって、Throwable オブジェクトをログ出力する際には、Log.d()/Log.v() メソッドのみを使用することを推奨する。

4.8.2.4 ログ出力には android.util.Log クラスのメソッドのみ使用する （推奨）

開発中にアプリが想定通りに動作していることを確認するために、System.out/err でログを出力することがあるだろう。もちろん System.out/err の print()/println() メソッドでも LogCat にログを出力することは可能だが、以下の理由からログ出力には android.util.Log クラスのメソッドのみを使用することを強く推奨する。

ログを出力するときは、一般には情報の緊急度に応じて出力メソッドを使い分け、出力を制御する。たとえば、深刻なエラー、警告、単なるアプリ情報通知などの区分が使われる。この区分を System.out/err に適用する手段の一つには、エラーと警告は System.err、それ以外は System.out で出力する方法がある。しかし、この場合、リリース時にも出力する必要のあるセンシティブでない情報 (運用ログ情報) とセンシティブな情報が含まれている可能性のある情報 (開発ログ情報) が同じメソッドによって出力されてしまう。よって、センシティブな情報を出力するコードを削除する際に、削除漏れが発生するおそれがある。

また、ログ出力に android.util.Log と System.out/err を使う場合は、android.util.Log のみを使う場合と比べて、ログ出力コードを削除する際に考慮することが増えるため、削除漏れなどのミスが生じるおそれがある。

上記のようなミスが生じる危険を減らすために、android.util.Log クラスのメソッドのみ使用することを推奨する。

4.8.3 アドバンスト

4.8.3.1 リリース版アプリにおけるログ出力の 2 つの考え方

リリース版 Android アプリにおけるログ出力の考え方には大きく分けて、一切ログ出力すべきではないという考え方と、後の解析のために必要な情報をログ出力すべきという考え方の 2 つがある。セキュリティ観点ではリリース版アプリでは一切ログ出力しないことが望ましい。しかし様々な理由によりリリース版アプリでもログ出力するケースがある。ここでは両者のそれぞれの考え方について述べる。

1 つ目は、リリース版アプリにおいてログ出力することにはあまり価値がなく、しかもセンシティブな情報を漏洩してしまうリスクがあるので、「一切ログ出力すべきではない」という考え方である。この考え方は、多くの Web アプリ運用環境などと違い、Android アプリ運用環境ではリリース後のアプリのログ情報を開発者が収集する手段が用意されていないことによるものである。この考え方に基づくと、開発中に使用したログ出力コードを最終版のソースコードから削除してリリース版アプリを作成するという運用がなされる。

2 つ目は、カスタマーサポート等でアプリの不具合解析を行う最終手段として、「後の解析のために必要な情報をログ出力すべき」という考え方である。この考え方に基づくと、リリース版アプリではセンシティブな情報を誤ってログ出力してしまわないよう細心の注意が必要となるため、サンプルコードセクションで紹介したような人為的ミスを排除する運用が必要となる。なお、下記の Google の Code Style Guideline も 2 つ目の考え方に基づいている。

Code Style Guidelines for Contributors / Log Sparingly https://source.android.com/setup/contribute/code-style#log-sparingly

4.8.3.2 ログレベルとログ出力メソッドの選択基準

Android の android.util.Log クラスには ERROR, WARN, INFO, DEBUG, VERBOSE の 5 段階のログレベルが定義されている。出力したいログ情報のログレベルに応じて、適切な android.util.Log クラスのログ出力メソッドを選択する必要がある。選択基準を 表 4.8.1 にまとめた。

表 4.8.1　ログレベルとログ出力メソッドの選択基準

ログレベル	メソッド	出力するログ情報の趣旨	アプリリリース時の注意
ERROR	Log.e	アプリが致命的な状況に陥ったときに出力するログ情報。	左記のログ情報はユーザーも参照することが 想定される情報であるため、開発版アプリと リリース版アプリの両方でログ出力されるべ き情報である。 そのためこのログレベルではセンシティブな 情報をログ出力してはならない。
WARN	Log.w	アプリが深刻な予期せぬ状況に遭遇 したときに出力するログ情報。	
INFO	Log.i	上記以外で、アプリの注目すべき状態の変化や結果を知らせる目的で出力 するログ情報。	
DEBUG	Log.d	アプリ開発時に特定のバグの原因究明のために一時的にログ出力したいプログラム内部の状態情報。	左記のログ情報はアプリ開発者専用の情報で あるため、リリース版アプリではログ出力さ れてはならない情報である。 開発版アプリではセンシティブな情報を出力 しても構わないが、リリース版アプリでは絶 対にセンシティブな情報をログ出力してはな らない。
VERBOSE	Log.v	以上のいずれにも該当しないログ情報。 アプリ開発者がさまざまな目的で出力 するログ情報が該当する。 サーバーとの通信データをダンプ出力 したい場合など。	

より詳細なログ出力の作法については下記 URL を参照すること。

Code Style Guidelines for Contributors / Log Sparingly https://source.android.com/setup/contribute/code-style#log-sparingly

4.8.3.3 DEBUG ログと VERBOSE ログは自動的に削除されるわけではない

Developer Reference の android.util.Log クラスの解説[*28] には次のような記載がある。

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.

開発者の中には、この文章から Log クラスの動作を次のように誤った解釈をしている人がいる。

- Log.v() 呼び出しはリリースビルド時にはコンパイルされず、VERBOSE ログが出力されることがなくなる
- Log.d() 呼び出しはコンパイルされるが、実行時には DEBUG ログが出力されることはない

しかし実際には Log クラスはこのようには動作せず、デバッグビルド、リリースビルドを問わず全てのログを出力して

[*28] https://developer.android.com/intl/ja/reference/android/util/Log.html

しまう。よく読んでみるとわかるが、この英文は Log クラスの動作について語っているのではなく、ログ情報とはこうあるべきということを説明しているだけである。

この記事のサンプルコードでは、ProGuard を使って上記英文のような動作を実現する方法を紹介している。

4.8.3.4 ログ情報の組み立て処理を削除する

下記ソースコードを ProGuard でリリースビルドして Log.d() を削除した場合、Log.d() の呼び出し処理（下記コードの 2 行目）は削除されるものの、その前段でセンシティブな情報を組み立てる処理（下記コードの 1 行目）は削除されないことに注意が必要である。

```
String debug_info = String.format("%s:%s", "センシティブな情報 1", "センシティブな情報 2");
if (BuildConfig.DEBUG) android.util.Log.d(TAG, debug_info);
```

上記ソースコードをリリースビルドした APK ファイルを逆アセンブルすると次のようになる。確かに Log.d() の呼び出し処理は存在しないが、” センシティブな情報 1” といった文字列定数定義と String#format() メソッドの呼び出し処理が削除されず残っていることが分かる。

```
const-string v1, "%s:%s"
const/4 v2, 0x2
new-array v2, v2, [Ljava/lang/Object;
const/4 v3, 0x0
const-string v4, "センシティブな情報 1"
aput-object v4, v2, v3
const/4 v3, 0x1
const-string v4, "センシティブな情報 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
```

実際には APK ファイルを逆アセンブルして、上記のようにログ出力情報を組み立てている箇所を発見するのは容易なことではない。しかし非常に機密度の高い情報を扱っているアプリにおいては、このような処理が APK ファイルに残ってしまってはならない場合もあり得る。

もし上記のようなログ出力情報の組み立て処理も削除してしまいたい場合には、次のように記述するとよい[*29]。リリースビルド時にはコンパイラの最適化処理によって、下記サンプルコードの処理は丸ごと削除される。

```
if (BuildConfig.DEBUG) {
    String debug_info = String.format("%s:%s", "センシティブな情報 1", "センシティブな情報 2");
    if (BuildConfig.DEBUG) android.util.Log.d(TAG, debug_info);
}
```

なお、下記ソースコードに ProGuard を適用した場合も、同様にログ情報の組み立て処理（”result:”+ value の部分）が残ってしまう。

[*29] 前述のサンプルコードを、条件式に BuildConfig.DEBUG を用いた if 文で囲った。Log.d() 呼び出し前の if 文は不要であるが、前述のサンプルコードと対比させるため、そのまま残した。

```
Log.d(TAG, "result:" + value);
```

この場合も下記のように対処すればよい。

```
if (BuildConfig.DEBUG) Log.d(TAG, "result:" + value);
```

4.8.3.5 Intent の内容が LogCat に出力される

Activity を利用する際に ActivityManager が Intent の内容を LogCat に出力するため、注意が必要である。「4.1.3.4. *Activity* 利用時のログ出力について」を参照すること。

4.8.3.6 System.out/err に出力されるログの抑制

System.out/err の出力先は LogCat である。System.out/err に出力されるのは、開発者がデバッグのために出力したログに限らない。例えば、次の場合、スタックトレースは System.err に出力される。

- Exception#printStackTrace() を使った場合
- 暗黙的に System.err に出力される場合（例外をアプリでキャッチしていない場合、システムが Exception#printStackTrace() に渡すため。）

スタックトレースにはアプリ固有の情報が含まれるため、例外は開発者が正しくハンドリングすべきである。

保険的対策として、System.out/err の出力先を LogCat 以外に変更する方法がある。以下に、リリースビルド時に System.out/err の出力先を変更し、どこにもログ出力しないようにする実装例を挙げる。ただし、この対応は System.out/err の出力先をアプリの実行時に一時的に書き換えるので、アプリやシステムの誤動作に繋がらないかどうかを充分に検討する必要がある。また、この対策はアプリ自身のプロセスには有効であるが、システムプロセスが生成するエラーログを抑制することはできない。すべてのエラーを抑制できるわけではないことに注意すること。

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;
```

(次のページに続く)

(前のページからの続き)

```
import java.io.IOException;
import java.io.OutputStream;
import java.io.PrintStream;

import android.app.Application;

public class OutputRedirectApplication extends Application {

    // どこにも出力しない PrintStream
    private final PrintStream emptyStream = new PrintStream(new OutputStream() {
        public void write(int oneByte) throws IOException {
            // do nothing
        }
    });

    @Override
    public void onCreate() {
        // リリースビルド時に System.out/err をどこにも出力しない PrintStream にリダイレクトする

        // System.out/err の本来のストリームを退避する
        PrintStream savedOut = System.out;
        PrintStream savedErr = System.err;

        // 一旦、System.out/err をどこにも出力しない PrintStream にリダイレクトする
        System.setOut(emptyStream);
        System.setErr(emptyStream);

        // デバッグ時のみ本来のストリームに戻す (リリースビルドでは下記 1 行が ProGuard により削除される)
        resetStreams(savedOut, savedErr);
    }

    // リリース時は ProGuard により下記メソッドがまるごと削除される
    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
```

(次のページに続く)

(前のページからの続き)

```
      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
# クラス名、メソッド名等の変更を防ぐ
-dontobfuscate

# リリースビルド時に Log.d()/v() の呼び出しを自動的に削除する
-assumenosideeffects class android.util.Log {
    public static int d(...);
    public static int v(...);
}

# リリースビルド時に resetStreams() を自動的に削除する
-assumenosideeffects class
    org.jssec.android.log.outputredirection.OutputRedirectApplication {
    private void resetStreams(...);
}
```

開発版アプリ（デバッグビルド）とリリース版アプリ（リリースビルド）の LogCat 出力の違いを 図 4.8.3 に示す。

図 4.8.3　System.out/err の開発版アプリとリリース版アプリの LogCat 出力の違い

4.9 WebView を使う

Web サイトや HTML ファイルを閲覧する機能を実装する方法として、WebView を使用することができる。WebView は HTML をレンダリングする、JavaScript を実行するなど、この目的のために有用な機能を提供する。

4.9.1 サンプルコード

WebView を使用することにより容易に Web サイト、HTML ファイル閲覧機能を実現することができるが、アクセスするコンテンツの特性によって WebView が抱えるリスクや適切な防衛手段が異なってくる。

特に気をつけなければいけないのは JavaScript の使用である。WebView のデフォルト設定では JavaScript の機能が無効になっているが、WebSettings#setJavaScriptEnabled() メソッドにより有効にすることが可能である。JavaScript を使用することでインタラクティブなコンテンツの表示が可能になるが、悪意のある第三者により端末の情報を取得される、あるいは端末を操作されるという被害が発生する可能性がある。

WebView を用いてコンテンツにアクセスするアプリを開発する際は、次の原則に従うこと[*30]。

(1) 自社が管理しているコンテンツにのみアクセスする場合に限り JavaScript を有効にしてよい

(2) 上記以外の場合には、JavaScript を有効にしてはならない

開発しているアプリがアクセスするコンテンツの特性を踏まえ、図 4.9.1 に従いサンプルコードを選択することが必要である。

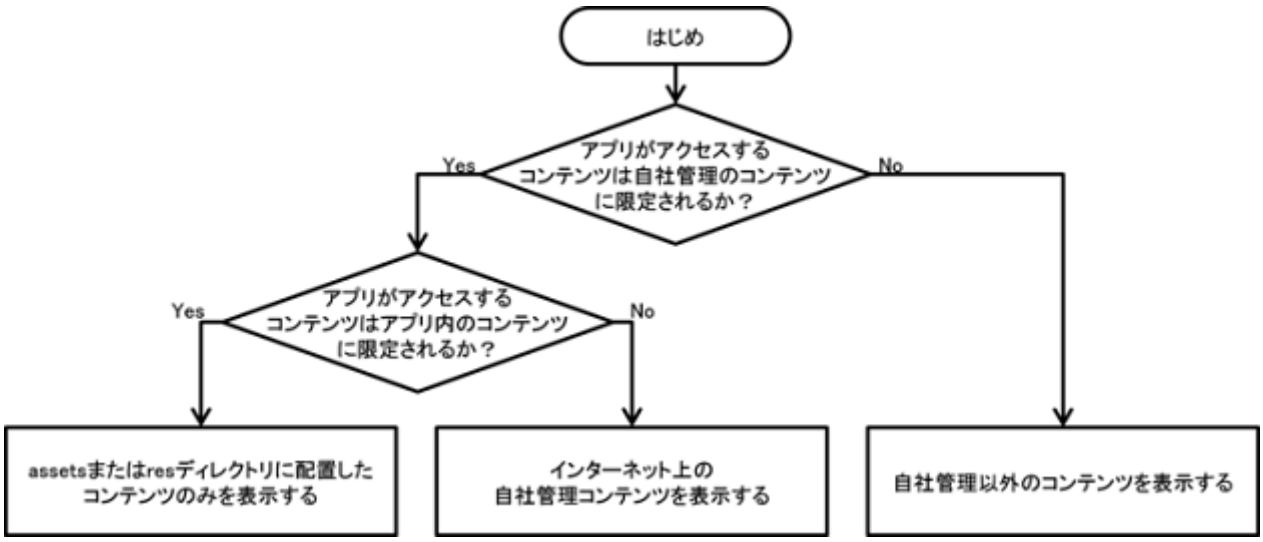

図 4.9.1　WebView のサンプルコードを選択するフローチャート

4.9.1.1 assets または res ディレクトリに配置したコンテンツのみを表示する

端末内のローカルコンテンツを WebView で表示するアプリに関しては、アプリの APK に含まれる assets あるいは res ディレクトリ内のコンテンツにアクセスする場合に限り JavaScript を有効にしてもよい。

以下に WebView を使用して assets ディレクトリ内にある HTML ファイルを表示するサンプルコードを示す。

[*30] 厳密に言えば安全性を保証できるコンテンツであれば JavaScript を有効にしてよい。自社管理のコンテンツであれば自社の努力で安全性を確保できるし責任も取れる。では信頼できる提携会社のコンテンツは安全だろうか？ これは会社間の信頼関係により決まる。信頼できる提携会社のコンテンツを安全であると信頼して JavaScript を有効にしてもよいが、万一の場合は自社責任も伴うため、ビジネス責任者の判断が必要となる。

ポイント：

1. assets と res ディレクトリ以外の場所に配置したファイルへのアクセスを禁止にする
2. JavaScript を有効にしてよい

WebViewAssetsActivity.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.webview.assets;

import android.app.Activity;
import android.os.Bundle;
import android.webkit.WebSettings;
import android.webkit.WebView;

public class WebViewAssetsActivity extends Activity {
    /**
     * assets 内のコンテンツを表示する
     */
    @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();

        // ★ポイント 1★ assets と res ディレクトリ以外の場所に配置したファイルへのアクセスを禁止にする
        webSettings.setAllowFileAccess(false);

        // ★ポイント 2★ JavaScript を有効にしてよい
        webSettings.setJavaScriptEnabled(true);
```

(次のページに続く)

(前のページからの続き)

```
        // assets 内に配置したコンテンツを表示する
        webView.loadUrl("file:///android_asset/sample/index.html");
    }
}
```

4.9.1.2 インターネット上の自社管理コンテンツを表示する

自社の管理するサービス上のコンテンツを表示する場合、サービス側、アプリ側の双方で適切な対策を施し、安全が確保できるならば、JavaScript を有効にしてもよい。

- サービス側の対策

図 4.9.2 に示すように、サービス側に用意するコンテンツは自社の管理していないコンテンツを参照してはならない。加えて、サービスに適切なセキュリティ対策が施されていることも必要である。その理由は、サービスを構成するコンテンツへの攻撃コードの埋め込みや改ざんを防止することにある。「4.9.2.1. *JavaScript* を有効にするのはコンテンツを自社が管理している場合に限定する（必須）」を参照すること。

- アプリ側の対策

次にアプリ側での対策を述べる。アプリ側では、接続先が自社管理サービスであることを確認することが必要である。そのために、通信プロトコルは HTTPS を使用し、証明書が信頼できる場合のみ接続するように実装する。

以下では、アプリ側での実装の例として、WebView を使って自社管理コンテンツを表示する Activity の例を示す。

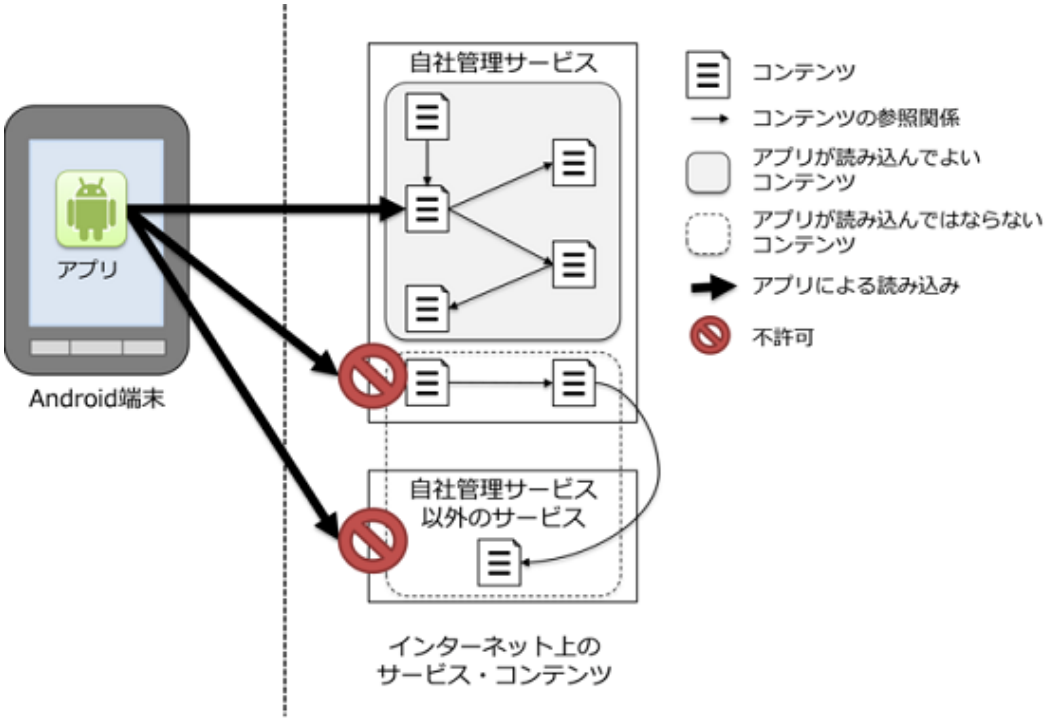

図 4.9.2　アプリが読み込んでよい自社管理コンテンツ

ポイント：

1. WebView の SSL 通信エラーを適切にハンドリングする

2. WebView の JavaScript を有効にしてもよい
3. WebView で表示する URL を HTTPS プロトコルだけに限定する
4. WebView で表示する URL を自社管理コンテンツだけに限定する

WebViewTrustedContentsActivity.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.webview.trustedcontents;

import android.app.Activity;
import android.app.AlertDialog;
import android.content.DialogInterface;
import android.net.http.SslCertificate;
import android.net.http.SslError;
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
```

(次のページに続く)

(前のページからの続き)

```
            public void onReceivedSslError(WebView view,
                                           SslErrorHandler handler, SslError error) {
                // ★ポイント 1★ WebView の SSL 通信エラーを適切にハンドリングする
                // SSL エラーが発生した場合には、SSL エラーが発生した旨をユーザに通知する
                AlertDialog dialog = createSslErrorDialog(error);
                dialog.show();

                // ★ポイント 1★ WebView の SSL 通信エラーを適切にハンドリングする
                // SSL エラーが発生した場合、有効期限切れなど証明書に不備があるか、
                // もしくは中間者攻撃を受けている可能性があるので、安全のために接続を中止する。
                handler.cancel();
            }
        });

        // ★ポイント 2★ WebView の JavaScript を有効にしてもよい
        // 以下のコードでは、loadUrl() で自社管理コンテンツを読みこむことを想定している。
        webView.getSettings().setJavaScriptEnabled(true);

        // ★ポイント 3★ WebView で表示する URL を HTTPS プロトコルだけに限定する
        // ★ポイント 4★ WebView で表示する URL を自社管理コンテンツだけに限定する
        webView.loadUrl("https://url.to.your.contents/");
    }

    private AlertDialog createSslErrorDialog(SslError error) {
        // ダイアログに表示するエラーメッセージ
        String errorMsg = createErrorMessage(error);
        // ダイアログの OK ボタン押下時の挙動
        DialogInterface.OnClickListener onClickOk = new DialogInterface.OnClickListener() {
            @Override
            public void onClick(DialogInterface dialog, int which) {
                setResult(RESULT_OK);
            }
        };
        // ダイアログの作成
        AlertDialog dialog = new AlertDialog.Builder
            (WebViewTrustedContentsActivity.this).setTitle("SSL 接続エラー")
            .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()
```

(次のページに続く)

(前のページからの続き)

```
            .append("サイトのセキュリティ証明書が信頼できません。接続を終了しました。\n\n エラーの原因\
↪n");
        switch (error.getPrimaryError()) {
        case SslError.SSL_EXPIRED:
            result.append("証明書の有効期限が切れています。\n\n 終了時刻=")
                .append(dateFormat.format(cert.getValidNotAfterDate()));
            return result.toString();
        case SslError.SSL_IDMISMATCH:
            result.append("ホスト名が一致しません。\n\nCN=")
                .append(cert.getIssuedTo().getCName());
            return result.toString();
        case SslError.SSL_NOTYETVALID:
            result.append("証明書はまだ有効ではありません\n\n 開始時刻=")
                .append(dateFormat.format(cert.getValidNotBeforeDate()));
            return result.toString();
        case SslError.SSL_UNTRUSTED:
            result.append("証明書を発行した認証局が信頼できません\n\n 認証局\n")
                .append(cert.getIssuedBy().getDName());
            return result.toString();
        default:
            result.append("原因不明のエラーが発生しました");
            return result.toString();
        }
    }
}
```

4.9.1.3 自社管理以外のコンテンツを表示する

自社で管理していないコンテンツを WebView で接続・表示する場合は、JavaScript を有効にしてはならない。攻撃者が用意したコンテンツに接続する可能性があるからである。

以下のサンプルコードは WebView を使用して自社管理以外のコンテンツを表示するアプリである。このアプリは、アドレスバーに入力した URL の指す HTML ファイルなどのコンテンツを読み込み、画面に表示する。安全の確保のために JavaScript を無効化しているほか、HTTPS で通信していて SSL エラーが発生した場合は接続を中止する実装となっている。SSL エラーは「4.9.1.2. インターネット上の自社管理コンテンツを表示する」と同様の方法によりハンドリングしている。HTTPS 通信についての詳細は、「5.4. *HTTPS* で通信する」を参照すること。

ポイント：

1. HTTPS 通信の場合には SSL 通信のエラーを適切にハンドリングする
2. JavaScript を有効にしない

WebViewUntrustActivity.java

```
/*
 * Copyright (C) 2012-2026 Japan Smartphone Security Association
```

(次のページに続く)

(前のページからの続き)

```
 *
 * 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.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 {
    /*
     * 自社管理以外のコンテンツを表示する (簡易ブラウザとして機能するサンプルプログラム)
     */

    private EditText textUrl;
    private Button buttonGo;
    private WebView webView;


    // この Activity が独自に URL リクエストをハンドリングできるようにするために定義
    private class WebViewUnlimitedClient extends WebViewClient {
```

(次のページに続く)

(前のページからの続き)

```
        @Override
        public boolean shouldOverrideUrlLoading(WebView webView, String url) {
            webView.loadUrl(url);
            textUrl.setText(url);
            return true;
        }

        // Web ページの読み込み開始処理
        @Override
        public void onPageStarted(WebView webview, String url, Bitmap favicon) {
            buttonGo.setEnabled(false);
            textUrl.setText(url);
        }

        // SSL 通信で問題があるとエラーダイアログを表示し、
        // 接続を中止する
        @Override
        public void onReceivedSslError(WebView webview,
                                       SslErrorHandler handler, SslError error) {
            // ★ポイント 1 ★ HTTPS 通信の場合には SSL 通信のエラーを適切にハンドリングする
            AlertDialog errorDialog = createSslErrorDialog(error);
            errorDialog.show();
            handler.cancel();
            textUrl.setText(webview.getUrl());
            buttonGo.setEnabled(true);
        }

        // Web ページの load が終わったら表示されたページの URL を EditText に表示させる
        @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());

        // ★ポイント 2 ★ JavaScript を有効にしない
        // デフォルトの設定で JavaScript 無効となっているが、明示的に無効化する
```

(次のページに続く)

(前のページからの続き)

```
        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) {
        // ダイアログに表示するエラーメッセージ
        String errorMsg = createErrorMessage(error);
        // ダイアログの OK ボタン押下時の挙動
        DialogInterface.OnClickListener onClickOk = new DialogInterface.OnClickListener() {
            @Override
            public void onClick(DialogInterface dialog, int which) {
                setResult(RESULT_OK);
            }
        };
        // ダイアログの作成
        AlertDialog dialog =
            new AlertDialog.Builder(WebViewUntrustActivity.this).setTitle("SSL 接続エラー")
                .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("サイトのセキュリティ証明書が信頼できません。接続を終了しました。\n\n エラーの原因\
↪n");
        switch (error.getPrimaryError()) {
        case SslError.SSL_EXPIRED:
            result.append("証明書の有効期限が切れています。\n\n 終了時刻=")
                .append(dateFormat.format(cert.getValidNotAfterDate()));
            return result.toString();
        case SslError.SSL_IDMISMATCH:
            result.append("ホスト名が一致しません。\n\nCN=")
                .append(cert.getIssuedTo().getCName());
            return result.toString();
        case SslError.SSL_NOTYETVALID:
```

(次のページに続く)

(前のページからの続き)

```
            result.append("証明書はまだ有効ではありません\n\n 開始時刻=")
                .append(dateFormat.format(cert.getValidNotBeforeDate()));
            return result.toString();
        case SslError.SSL_UNTRUSTED:
            result.append("証明書を発行した認証局が信頼できません\n\n 認証局\n")
                .append(cert.getIssuedBy().getDName());
            return result.toString();
        default:
            result.append("原因不明のエラーが発生しました");
            return result.toString();
        }
    }
}
```

4.9.2 ルールブック

WebView を使用する際には以下のルールを守ること。

1. *JavaScript* を有効にするのはコンテンツを自社が管理している場合に限定する（必須）
2. 自社管理サービスとの通信には *HTTPS* を使用する（必須）
3. *Intent* 経由など、他から受け取った *URL* が想定された *URL* か確認する（必須）
4. *SSL* 通信のエラーを適切にハンドリングする（必須）

4.9.2.1 JavaScript を有効にするのはコンテンツを自社が管理している場合に限定する（必須）

WebView を用いてコンテンツやサービスにアクセスするアプリを開発する際に、セキュリティの面で最も注意しなければならない点は JavaScript を有効にするかどうかである。原則的には、自社が管理しているサービスにのみアプリがアクセスする場合に限り JavaScript を有効にしてもよい。しかし、そうでないサービスにアクセスする可能性が少しでもある場合には、JavaScript を有効にしてはならない。

4.9.2.2 自社で管理しているサービス

自社で作成あるいは、運用、管理に責任を持つサービスは、自社が安全を保証できる。例として、自社管理サーバー上の自社開発コンテンツにアプリがアクセスする場合を考える。各コンテンツがサーバー内部のコンテンツのみを参照しており、かつ、自社管理サーバーに対して適切なセキュリティ対策が施されているならば、このサービスは自社以外が内容を書き換えていることはないとみなせる。この場合、自社管理サービスにアクセスするアプリの JavaScript を有効にしてもよい。「4.9.1.2. インターネット上の自社管理コンテンツを表示する」を参照すること。また、他のアプリによる書き換えが不可能な端末内コンテンツ (APK の assets または res ディレクトリ内に配置されたコンテンツやアプリディレクトリ下のコンテンツ) にアクセスするアプリの場合も同様に考え、JavaScript を有効にしてもよい。「4.9.1.1. *assets* または *res* ディレクトリに配置したコンテンツのみを表示する」を参照すること。

4.9.2.3 自社で管理していないサービス

自社で管理していないコンテンツ・サービスは自社が安全を保証できると考えてはならない。それゆえ、アプリの JavaScript を無効にしなければならない。「4.9.1.3. 自社管理以外のコンテンツを表示する」を参照すること。加えて、SD カードのような端末の外部記憶装置に配置されたコンテンツは他のアプリによる書き換えが可能なので、自社が管理しているとは言えない。そのようなコンテンツにアクセスするアプリについても JavaScript を無効にしなければならない。

4.9.2.4 自社管理サービスとの通信には **HTTPS** を使用する（必須）

自社管理サービスにアクセスするアプリは、悪意ある第三者によるサービスのなりすましによる被害を防ぎ、対象サービスへ確実に接続する必要がある。そのためには、サービスとの通信に HTTPS を使用する。

詳細は「4.9.2.6. *SSL* 通信のエラーを適切にハンドリングする（必須）」、「5.4. *HTTPS* で通信する」を参照すること。

4.9.2.5 **Intent** 経由など、他から受け取った **URL** が想定された **URL** か確認する（必須）

他のアプリから Intent を受信し、その Intent のパラメータで渡された URL を WebView に表示する実装が多くのアプリで見られる。このとき渡された URL を、想定された URL か確認することなく表示する場合、フィッシングサイトなどの悪意のある Web ページも WebView 上で表示されてしまう。この実装の問題点は、安全を保証できない不特定の URL を表示してしまうことであり、これは WebView の JavaScript が無効であっても被害が生じる可能性がある。

Intent のパラメータで渡された URL が想定された URL か確認する方法としては、あらかじめアプリ内に表示したい URL のホワイトリストを保持しておく方法がある。このホワイトリストと照合して合致した URL のみを WebView 上に表示することで安全を確保することができる。なお、ホワイトリストに登録する URL は HTTPS でなければならないことにも注意が必要である。

WebViewActivity.java

```
    // リソースファイルからホワイトリストを取得する
    String[] allowList = getResources().getStringArray(R.array.allow_url_list);

    // Intent から受け取った URL のドメインがホワイトリストに含まれているか確認する
    Uri uri = Uri.parse("Intent から受け取った URL");
    for (String str : allowList) {
        if (uri.getScheme().equals("https") && uri.getHost().equals(str)) {
            webView.loadUrl(uri.toString());
    }
```

渡された URL を JavaScript が有効な WebView で表示したい場合は、その URL が自社管理コンテンツであることを追加で確認する必要がある。

サンプルコード「4.9.1.2. インターネット上の自社管理コンテンツを表示する」では、固定 URL 文字列定数で自社管理コンテンツを指定することで、WebView で表示するコンテンツを自社管理コンテンツに限定し安全を確保している。

4.9.2.6 **SSL** 通信のエラーを適切にハンドリングする（必須）

HTTPS 通信で SSL エラーが発生した場合は、エラーが発生した旨をダイアログ表示するなどの方法でユーザーに通知して、通信を終了しなければならない。

SSL エラーの発生は、サーバー証明書に不備がある可能性、あるいは中間者攻撃を受けている可能性を示唆する。しか

し、WebView には、サービスとの通信時に発生した SSL エラーに関する情報をユーザーに通知する仕組みが備わっていない。そこで、SSL エラーが発生した場合にはその旨をダイアログなどで表示することで、脅威にさらされている可能性があることをユーザーに通知する必要がある。エラー通知の例は、「4.9.1.2. インターネット上の自社管理コンテンツを表示する」のサンプルコードあるいは「4.9.1.3. 自社管理以外のコンテンツを表示する」のサンプルコードを参照すること。

また、エラーの通知に加えて、アプリはサービスとの通信を終了しなければならない。特に、次のような実装を行ってはならない。

- 発生したエラーを無視してサービスとの通信を継続する
- HTTP などの非暗号化通信を使ってサービスと改めて通信する

HTTP 通信/HTTPS 通信の詳細は「5.4. *HTTPS* で通信する」を参照すること。

SSL エラーが発生した際には対象のサーバーと接続を行わないことが WebView のデフォルトの挙動である。よって、WebView のデフォルトの挙動に SSL エラーの通知機能を実装することで適切に通信エラーを取り扱うことができる。

4.9.3 アドバンスト

4.9.3.1 file スキームに起因する問題について

WebView をデフォルト設定で使用している場合、file スキームを利用してアクセスすると当該アプリがアクセス可能なすべてのファイルにアクセスすることが可能になる。この動作を悪用された場合、例えば、JavaScript から file スキーム使ったリクエストすることで、アプリの専用フォルダに保存したファイル等を攻撃者に取得されてしまう可能性がある。

対策としては、「4.9.2.1. *JavaScript* を有効にするのはコンテンツを自社が管理している場合に限定する（必須）」で解説した通り、自社で管理していないコンテンツ・サービスにアクセスする可能性がある場合は JavaScript を無効にする。この対策により意図しない file スキームによるリクエストが送信されないようにする。

また、Android 4.1（API Level 16）以降の場合、setAllowFileAccessFromFileURLs() および setAllowUniversalAccessFromFileURLs() を利用することで file スキームによるアクセスを禁止することができる。

file スキームの無効化

```
webView = (WebView) findViewById(R.id.webview);
webView.setWebViewClient(new WebViewUnlimitedClient());
WebSettings settings = webView.getSettings();
settings.setAllowUniversalAccessFromFileURLs(false);
settings.setAllowFileAccessFromFileURLs(false);
```

4.9.3.2 Web Messaging 利用時の送信先オリジン指定について

Android 6.0(API Level 23) において、HTML5 Web Messaging を実現するための API が追加された。Web Messaging は異なるブラウジング・コンテキスト間でデータを送受信するための仕組みであり、HTML5 で定義されている[*31]。

WebView クラスに追加された postWebMessage() は Web Messaging で定義されている Cross-domain messaging によるデータ送信を処理するメソッドである。このメソッドは第 1 引数で指定されたメッセージオブジェクトを WebView に読み込んでいるブラウジング・コンテキストに対して送信するのだが、その際第 2 引数として送信先のオリジンを指定す

[*31] https://www.w3.org/TR/webmessaging/ <https://www.w3.org/TR/webmessaging/

る必要がある。指定されたオリジン[*32] が送信先コンテキストのオリジンと一致しない限りメッセージは送信されない。送信先オリジンを制限することで、意図しない送信先にメッセージを渡してしまうことを防いでいるのである。

ただし、postWebMessage() メソッドではオリジンとしてワイルドカードを指定できることに注意が必要である[*33]。ワイルドカードを指定するとメッセージの送信先オリジンがチェックされず、どのようなオリジンに対してもメッセージを送信してしまう。もし WebView に悪意のあるコンテンツが読み込まれている状況でオリジンの制限なしに重要なメッセージを送信してしまうと何らかの被害につながる可能性も生じる。WebView を用いて Web messaging を行う際は、postWebMessage() メソッドに特定のオリジンを明示的に指定するべきである。

4.9.3.3 WebView の Safe Browsing について

Safe Browsing とは Google が提供するサービスで、マルウェアページやフィッシングサイトなどの安全でない Web ページにユーザーがアクセスしようとした際に警告ページを表示させる仕組みである。

図 4.9.3　Chrome for Android で安全でない Web ページにアクセスした際に表示される警告ページ

現在、Chrome for Android などのブラウザアプリに限らず、アプリで使われる WebView においても Safe Browsing の機能を利用することが可能となっている。しかし、端末の Android OS バージョンにより WebView として用いられるコンポーネントが異なり、Safe Browsing の対応状況が異なるため注意が必要である。端末の Android OS バージョンと標準 WebView の対応と Safe Browsing の対応を次の表にまとめた。

[*32] オリジンとは、URL のスキーム、ホスト名、ポート番号の組み合わせのこと。詳細な定義は https://tools.ietf.org/html/rfc6454 を参照。
[*33] Uri.EMPTY および Uri.parse("") がワイルドカードとして機能する (2016 年 9 月 1 日版執筆時)

表 4.9.1　端末の Android OS バージョンと標準 WebView の対応

Android OS バージョン	**Android** 標準 **WebView**	**OS** との関係	**Safe Browsing** 対応
Android 7.0 以降	Chrome for Android (Chromium ベース)	独立	○
Android 5.0 - 6.0	Android System WebView (Chromium ベース)	独立	○
Android 4.4	OS 組み込み WebView (Chromium ベース)	一体	×
Android 4.3 以前	OS 組み込み WebView	一体	×

Android 4.3 (API Level 18) 以前は、Safe Browsing 機能が搭載されていない WebView が OS に組み込まれていたが、Android 4.4 (API Level 19) にて Safe Browsing 機能を持っているものに変更された。しかし、バージョンが古く、アプリの WebView での Safe Browsing 機能の利用には対応していないため注意すること。

Safe Browsing 機能をアプリの WebView で利用できるのは、WebView が OS から切り離されアプリとして更新されるようになった Android 5.0 (API Level 21) からである。

WebView 66 以降では Safe Browsing がデフォルトで有効になっており、アプリ側で特別何か設定を行う必要はない。しかし、ユーザーが WebView の更新を行っていなかった場合や、開発者向けオプション「WebView の実装」項目で標準 WebView をデフォルトのものから変更している場合、その WebView のバージョンによっては Safe Browsing がデフォルトで有効とならない可能性が考えられる。そのため、Safe Browsing を利用するのならば下記のように Safe Browsing を明示的に有効化しておくべきである。

AndroidManifest.xml での Safe Browsing の有効化設定

```
<?xml version="1.0" encoding="utf-8"?>
<manifest package="...">
    <application>
        ...

        <meta-data
            android:name="android.webkit.WebView.EnableSafeBrowsing"
            android:value="true" />
    </application>
</manifest>
```

また、Android 8.0 (API Level 26) において、Safe Browsing に関する API が幾つか追加された。

WebSettings クラスに追加された setSafeBrowsingEnabled(boolean enabled) は WebView のインスタンス単位で動的に有効・無効を設定するメソッドである。Android 8.0 (API Level 26) 未満では Safe Browsing 機能の有効・無効を AndroidManifest で設定していたが、これではアプリの全 WebView に対する一括の設定しかできなかった。setSafeBrowsingEnabled(boolean enabled) を用いることで WebView のインスタンス単位で動的に有効・無効を切り替えることが可能となった。

```
if (url == IN_HOUSE_MANAGEMENT_CONTENT_URL) {
    // 例) 自社管理コンテンツが Safe Browsing で検出されてしまうので一時的に無効化する
    webView.getSettings().setSafeBrowsingEnabled(false);
} else {
```

(次のページに続く)

(前のページからの続き)

```
    // 通常は有効化しておくべき
    webView.getSettings().setSafeBrowsingEnabled(true);
}
```

Android 8.1 (API Level 27) においても、Safe Browsing に関するクラス・API が追加された。これにより Safe Browsing の初期化処理の記述や、安全でない Web ページにアクセスした際の挙動の設定、特定のサイトを Safe Browsing の対象から外すためのホワイトリストの設定などが可能となった。

WebView クラスに追加された startSafeBrowsing() はアプリ内 WebView で使われる WebView コンポーネントの Safe Browsing 機能初期化処理を呼び出すメソッドである。第 2 引数で渡す Callback オブジェクトに初期化の結果が渡されるので、初期化が失敗し Callback オブジェクトに false が渡された場合、WebView を無効化する・URL をロードしないなどの対応が推奨される。

```
// Android 8.1 未満の端末ではサポートされないため、実際の実装時には端末の Android OS バージョンで処理を分ける必要がある
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...");
            // 初期化処理が失敗したため、Safe Browsing がうまく動作しない可能性がある
            // この場合 WebView を無効化することが望ましい
        }
    }
});
```

同じく WebView クラスに追加された setSafeBrowsingWhitelist() は Safe Browsing の対象から外したいホスト名・IP アドレスをホワイトリスト形式で設定するメソッドである。Safe Browsing の対象から外したい Host 名・IP アドレスのリストを引数で渡すとそれらにアクセスする際に Safe Browsing による検証が行われなくなる。

```
// Safe Browsing 機能を適用させない Host 名・IP アドレスのホワイトリストを設定する
// 例) 自社管理コンテンツが Safe Browsing で検出されてしまうのでホワイトリストに登録する
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());
        }
    });
```

WebClient クラスに追加された onSafeBrowsingHit() は、Safe Browsing が有効化された WebView でアクセスした URL が安全でない Web ページだと判定された際に Callback される Callback 関数である。第 1 引数に安全でない Web ページ

にアクセスした WebView の Object、第 2 引数に WebResourceRequest、第 3 引数に threat の種類、第 4 引数に判定に対する挙動を設定するための SafeBrowsingResponse オブジェクトが渡される。

SafeBrowsingResponse オブジェクトを使って選択できる挙動には、以下の 3 つがある。

- backToSafety(boolean report): 警告は表示せず前のページに戻る (前のページがない場合、blank ページを表示)
- proceed(boolean report): 警告を無視して Web ページを表示
- showInterstitial(boolean allowReporting): 警告ページを表示 (デフォルトの挙動)

backToSafety() および proceed() は引数で Google にレポートを送信するか否かを引数で設定でき、showInterstitial() は「Google にレポートを送信するかの選択を行うチェックボックス」を表示させるかを引数で設定できる。

```
public class MyWebViewClient extends WebViewClient {

    // Safe Browsing 機能が有効化された状態で安全でない Web ページにアクセスすると Callback される
    @Override
    public void onSafeBrowsingHit(WebView view, WebResourceRequest request,
                                  int threatType, SafeBrowsingResponse callback) {
        // 「Google にレポートを送信するか」を選択するチェックボックスを持った警告ページを表示する (推奨)
        callback.showInterstitial(true);
        // 警告ページは表示せず安全なページに戻し、Google にレポートを送信する (推奨)
        callback.backToSafety(true);
        // 警告を無視しページにアクセスし、Google にレポートを送信する (非推奨)
        callback.proceed(false);
    }
}
```

これらのクラス・API には対応する Android Support Library が無い。したがってこれらのクラス・API を利用するアプリを API Level 26 または 27 未満の端末で動作させたい場合はバージョンによって処理を分けるなどの措置が必要である。

WebView で Safe Browsing を利用し、安全でない Web ページへのアクセスをハンドリングするためのサンプルコードを以下に示す。

```
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
```

(次のページに続く)

(前のページからの続き)

```
        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>

    <!-- アプリプロセス内の WebView の Safe Browsing 機能を有効化する -->
    <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.
 * 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;
```

(次のページに続く)

(前のページからの続き)

```
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;
        // Android 8.1 未満の端末ではサポートされないため、実際の実装時には
        // 端末の Android OS バージョンで処理を分ける必要がある
        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...");
                    // 初期化処理が失敗したため、Safe Browsing がうまく動作しない可能性がある
                    // この場合 URL をロードしないことが望ましい
                }
            }
        });
    }

    View.OnClickListener setWhiteList = new View.OnClickListener() {
        @Override
        public void onClick(View view) {
            // Safe Browsing 機能を適用させない Host 名・IP アドレスのホワイトリストを設定する
            WebView.setSafeBrowsingWhitelist(new ArrayList<>(Arrays.asList(
↪"testsafebrowsing.appspot.com")),
                new ValueCallback<Boolean>() {
                @Override
                public void onReceiveValue(Boolean aBoolean) {
```

(次のページに続く)

(前のページからの続き)

```
                    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 {

    // Safe Browsing 機能が有効化された状態で安全でない Web ページにアクセスすると Callback される
    @Override
```

(次のページに続く)

(前のページからの続き)

```
    public void onSafeBrowsingHit(WebView view, WebResourceRequest request,
                                  int threatType, SafeBrowsingResponse callback) {
        // 警告ページは表示せず安全なページに戻す
        callback.backToSafety(true);
        Toast.makeText(view.getContext(),
            "アクセスしようとした Web ページはマルウェアサイトの疑いがあるため、安全なページに戻ります。",
            Toast.LENGTH_LONG).show();
    }
}
```

4.10 Notification を使用する

Android にはエンドユーザーへのメッセージを通知する Notification 機能がある。Notification を使うと、画面上部のステータスバーと呼ばれる領域に、アイコンやメッセージを表示することができる。

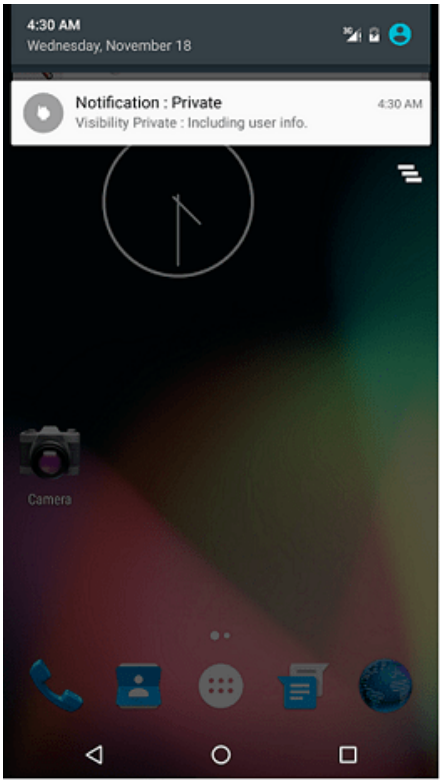

図 4.10.1　Notifcation の表示例

Notification の通知機能は、Android 5.0(API Level 21) で強化され、アプリやユーザー設定によって、画面がロックされている状態であっても Notification による通知を表示することが可能になった。ただし、Notification の使い方を誤ると、端末ユーザー本人にのみ見せるべきプライベートな情報が第三者の目に触れる恐れがある。したがって、プライバシーやセキュリティを考慮して適切に実装を行うことが重要である。

なお、Visibility が取り得る値と Notification の振る舞いは以下の通りである。

表 4.10.1　Visibility が取り得る値と Notification の振る舞い

Visibility の値	**Notification** の振る舞い
Public	すべてのロック画面上で Notification が表示される
Private	すべてのロック画面上で Notifcation が表示されるが、パスワード等で保護されたロック画面（セキュアロック）上では、Notification のタイトルやテキスト等が隠される（プライベート情報が隠された公開可能な文に置き換わる）
Secret	パスワード等で保護されたロック画面（セキュアロック）上では、Notification が表示されなくなる（セキュアロック以外のロック画面では Notification は表示される）

4.10.1 サンプルコード

Notification に端末ユーザーのプライベートな情報を含む場合、プライベート情報を取り除いた通知を画面ロック時の表示用に作成し、加えておくこと。

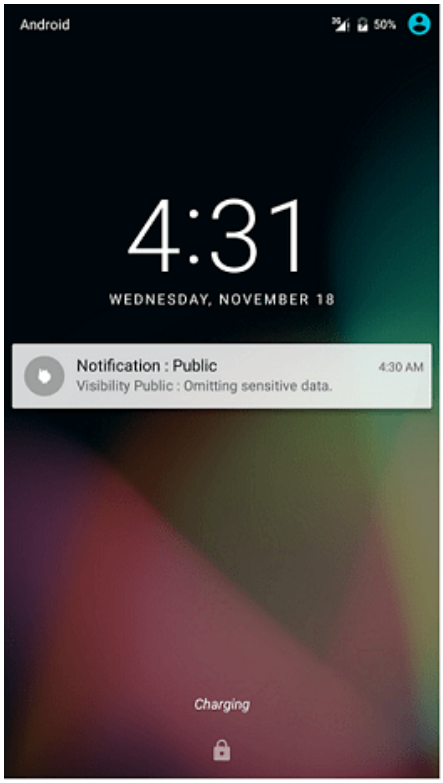

図 4.10.2　ロック画面上の Notification

プライベート情報を含んだ通知を行うサンプルコードを以下に示す。

ポイント：

1. プライベート情報を含んだ通知を行う場合は、公開用（画面ロック時の表示用）の Notification を用意する
2. 公開用（画面ロック時の表示用）の Notification にはプライベート情報を含めない
3. Visibility を明示的に Private に設定して、Notification を作成する
4. Visibility が Private の場合、プライベート情報を含めて通知してもよい

VisibilityPrivateNotificationActivity.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.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 {
    /**
     * Private な Notification を表示する
     */
    private final int mNotificationId = 0;

    public static final String DEFAULT_CHANNEL = "default_channel";
    public static final String SENDER_NAME = "Sender Name";
```

(次のページに続く)

(前のページからの続き)

```
    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) {
            // api level 26 からは Notification に channel が必要になる
            NotificationChannel default_channel =
                new NotificationChannel(DEFAULT_CHANNEL,
                                        getString(R.string.notification_channel_default),
                                        NotificationManager.IMPORTANCE_DEFAULT);

            NotificationManager notificationManager =
                (NotificationManager) this.getSystemService(Context.NOTIFICATION_SERVICE);
            notificationManager.createNotificationChannel(default_channel);
        }
    }

    public void onSendNotificationClick(View view) {

        // ★ポイント 1★  プライベート情報を含んだ通知を行う場合は、公開用（画面ロック時の表示用）の
Notification を用意する
        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);
        // ★ポイント 2★ 公開用（画面ロック時の表示用）の Notification にはプライベート情報を含めない
        publicNotificationBuilder.setContentText("Visibility Public : Omitting sensitive␣
↪data.");
        publicNotificationBuilder.setSmallIcon(R.drawable.ic_launcher);

        Notification publicNotification = publicNotificationBuilder.build();
```

(次のページに続く)

(前のページからの続き)

```
        // プライベート情報を含む Notification を作成する
        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");
        }

        // ★ポイント 3★ Visibility を明示的に Private に設定して、Notification を作成する
        if (Build.VERSION.SDK_INT >= 21)
            privateNotificationBuilder.setVisibility(Notification.VISIBILITY_PRIVATE);
        // ★ポイント 4★ Visibility が Private の場合、プライベート情報を含めて通知してもよい
        privateNotificationBuilder.setContentText("Visibility Private : Including user␣
↪info.");
        privateNotificationBuilder.setSmallIcon(R.drawable.ic_launcher);
        // Visibility が Private の Notification を利用する場合、
        // Visibility を Public にした公開用の Notification を合わせて設定する
        if (Build.VERSION.SDK_INT >= 21)
            privateNotificationBuilder.setPublicVersion(publicNotification);

        if (Build.VERSION.SDK_INT >= 28) {
            // Notification.MessagingStyle.Message.setData() により
            // リソースの sample_picture.png を Notification に表示する
            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();

            // Notification.MessagingStyle.Message を用意し setData() で画像を設定する
            Notification.MessagingStyle.Message message =
                new Notification.MessagingStyle.Message("Sample Picture", 0, sender)
                    .setData("image/png", imageUri);

            // Notification.MessagingStyle を用意し画像を設定した
            // Notification.MessagingStyle.Message を設定する
```

(次のページに続く)

(前のページからの続き)

```
            Notification.MessagingStyle message_style =
                new Notification.MessagingStyle(sender)
                    .addMessage(message);

            // Notification.MessagingStyle を Notification に設定する
            privateNotificationBuilder.setStyle(message_style);
        }

        if (Build.VERSION.SDK_INT >= 28) {
            // RemoteInput.Builder.setChoices() により Notification にリプライの選択肢を表示する
            Intent intent =
                new Intent( getApplicationContext(), NotificationReceiver.class);

            PendingIntent pendingIntent =
                PendingIntent.getBroadcast(this, 0, intent, FLAG_MUTABLE);

            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)
                    .setSemanticAction(Notification.Action.SEMANTIC_ACTION_REPLY)
                    .build();

            privateNotificationBuilder.addAction(actionReply);
        }

        Notification privateNotification = privateNotificationBuilder.build();

        // 本サンプルでは実装していないが、Notification では setContentIntent(PendingIntent␣
↪intent)
        // を使い Notification をクリックした際に Intent が送信されるように実装することが多い。
        // このときに設定する Intent は、呼び出すコンポーネントの種類に合わせて、
        // 安全な方法で呼び出すことが必要である（例えば、明示的 Intent を使うなど）
        // 各コンポーネントの安全な呼び出し方法は以下の項目を参照のこと
        // 4.1. Activity を作る・利用する
        // 4.2. Broadcast を受信する・送信する
        // 4.4. Service を作る・利用する

        NotificationManager notificationManager =
```

(次のページに続く)

(前のページからの続き)

```
                (NotificationManager) this.getSystemService(Context.NOTIFICATION_SERVICE);
        notificationManager.notify(mNotificationId, privateNotification);
    }
}
```

4.10.2 ルールブック

Notification を利用する際には以下のルールを守ること。

1. *Visibility* の設定に依らず、*Notification* にはセンシティブな情報を含めない（プライベート情報は例外）（必須）
2. *Visibility Public* の *Notification* には プライベート情報を含めない（必須）
3. （特に *Visibility Private* にする場合）*Visibility* は明示的に設定する（必須）
4. *Visibility* が *Private* の *Notification* を利用する場合、*Visibility* を *Public* にした公開用の *Notification* を併せて設定する （推奨）

4.10.2.1 Visibility の設定に依らず、Notification にはセンシティブな情報を含めない（プライベート情報は例外）（必須）

Android4.3(API Level 18) 以降の端末では、設定画面からユーザーが Notification の読み取り許可をアプリに与えることができる。許可されたアプリは、全ての Notification の情報を読み取ることが可能になるため、センシティブな情報を Notification に含めてはならない（ただし、プライベート情報は Visibility の設定によっては Notification に含めて良い）。

Notification に含まれた情報は、通常は Notification を送信したアプリを除き、他のアプリから読み取ることはできない。しかし、ユーザーが明示的に許可を与えることで、ユーザーが指定したアプリは全ての Notification の情報を読み取ることが可能になる。ユーザーが許可を与えたアプリのみが Notification の情報を読み取れることから、ユーザー自身のプライベート情報を Notification に含めることは問題ない。一方で、ユーザーのプライベート情報以外のセンシティブな情報（例えば、アプリ開発者のみが知り得る機密情報）を Notification に含めると、ユーザー自身が Notification に含まれた情報を読みとろうとして Notification への閲覧をアプリに許可する可能性があるため、利用者のプライベート情報以外のセンシティブな情報を含めることは問題となる。

具体的な方法と条件は、「4.10.3.1. ユーザー許可による *Notification* の閲覧について」を参照の事。

4.10.2.2 Visibility Public の Notification には プライベート情報を含めない（必須）

Visibility が Public に設定された Notification によって通知を行う場合、ユーザーのプライベート情報を Notification に含めてはならない。Visibility が Public に設定された Notification は、画面ロック中にも Notification の情報が表示され、端末に物理的に接近できる第三者がプライベート情報を盗み見るリスクにつながるためである。

VisibilityPrivateNotificationActivity.java

```
    // 公開用（画面ロック時の表示用）の センシティブな情報を持たない Notification を用意する
    Notification.Builder publicNotificationBuilder =
        new Notification.Builder(this).setContentTitle("Notification : Public");

    publicNotificationBuilder.setVisibility(Notification.VISIBILITY_PUBLIC);
```

(次のページに続く)

(前のページからの続き)

```
    // 公開用（画面ロック時の表示用）の Notification にはプライベート情報を含めない
    publicNotificationBuilder.setContentText("Visibility Public : センシティブな情報は含めずに通
知");
    publicNotificationBuilder.setSmallIcon(R.drawable.ic_launcher);
```

プライベート情報の典型例としては、ユーザー宛てに送信されたメールやユーザーの位置情報など、「5.5. プライバシー情報を扱う」で言及されている情報が挙げられる。

4.10.2.3 （特に **Visibility Private** にする場合）**Visibility** は明示的に設定する（必須）

「4.10.2.2. *Visibility Public の Notification には プライベート情報を含めない（必須）*」の通り、Android 5.0(API Level 21) 以降の端末では、画面ロック中にも Notification が表示されるため、Visibility の設定が重要であり、デフォルト値に頼らず明示的に設定すること。

現状では、Notification の Visibility のデフォルト値は Private に設定されており、明示的に Public を指定しない限りプライベート情報が盗み見られるリスクは発生しない。しかし、Visibility のデフォルト値が将来変更になる可能性もあり、含める情報の取り扱いを常に意識するためにも、たとえ Visiblity を Private にする場合であっても、Notification の Visibility は明示的に設定することを必須としている。

```
VisibilityPrivateNotificationActivity.java
    // プライベート情報を含む Notification を作成する
    Notification.Builder priavteNotificationBuilder =
        new Notification.Builder(this).setContentTitle("Notification : Private");

    // ★ポイント 3★ Visibility を明示的に Private に設定して、Notification を作成する
    priavteNotificationBuilder.setVisibility(Notification.VISIBILITY_PRIVATE);
```

4.10.2.4 **Visibility** が **Private** の **Notification** を利用する場合、**Visibility** を **Public** にした公開用の **Notification** を併せて設定する （推奨）

Visibility が Private に設定された Notification を使って通知する場合、画面ロック中に表示される情報を制御するため、Visibility を Public にした公開用の Notification を併せて設定することが望ましい。

Visibility が Private に設定された Notification に公開用の Notification を設定しない場合、画面ロック中にはシステムで用意されたデフォルトの文言が表示されるためセキュリティ上の問題はない。しかし、Notification に含める情報の取り扱いを常に意識するためにも、公開用の Notification を明示的に用意し設定することを推奨する。

```
VisibilityPrivateNotificationActivity.java
    // プライベート情報を含む Notification を作成する
    Notification.Builder privateNotificationBuilder =
        new Notification.Builder(this).setContentTitle("Notification : Private");

    // ★ポイント 3★ Visibility を明示的に Private に設定して、Notification を作成する
    if (Build.VERSION.SDK_INT >= 21)
        privateNotificationBuilder.setVisibility(Notification.VISIBILITY_PRIVATE);
    // ★ポイント 4★ Visibility が Private の場合、プライベート情報を含めて通知してもよい
```

(次のページに続く)

(前のページからの続き)

```
    privateNotificationBuilder.setContentText("Visibility Private : Including user info.");
    privateNotificationBuilder.setSmallIcon(R.drawable.ic_launcher);
    // Visibility が Private の Notification を利用する場合、
    // Visibility を Public にした公開用の Notification を合わせて設定する
    if (Build.VERSION.SDK_INT >= 21)
        privateNotificationBuilder.setPublicVersion(publicNotification);
```

4.10.3 アドバンスト

4.10.3.1 ユーザー許可による Notification の閲覧について

「4.10.2.1. *Visibility* の設定に依らず、*Notification* にはセンシティブな情報を含めない（プライベート情報は例外）（必須）」で述べたように、Android4.3(API Level 18) 以降の端末では、ユーザーが許可を与えた場合、指定されたアプリは全ての Notification の情報を読み取ることが可能になる。ただし、ユーザー許可の対象となるためには、アプリが NotificationListenerService を継承した Service を実装しておく必要がある。

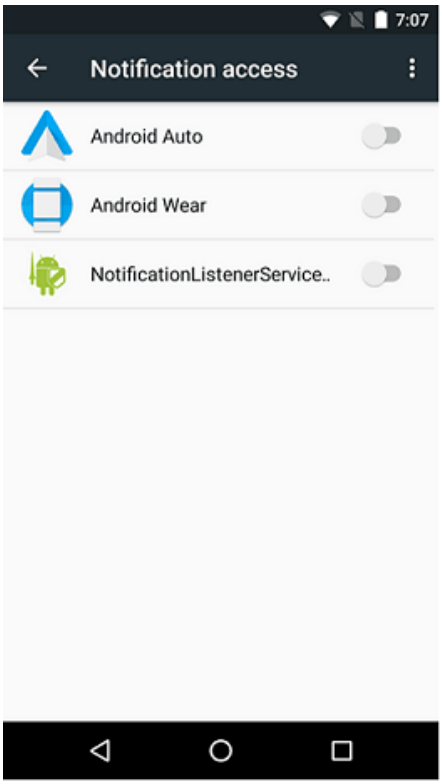

図 4.10.3　Notification の読み取りを設定する「通知へのアクセス」画面

NotificationListenerService を使ったサンプルコードを以下に示す[*34]。

```
AndroidManifest.xml
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
          >
```

(次のページに続く)

[*34] 第 12 版執筆時の環境における動作確認では、Android Studio 4.0.1 付属のエミュレータ Pixel 3 API 30 ではサンプルが動作しないことが判明している。物理端末 Google Pixel 3 では問題なく動作することを確認しているため、サンプルコード実行時には留意されたい。

(前のページからの続き)

```
  <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 {
```

(次のページに続く)

(前のページからの続き)

```
    @Override
    public void onNotificationPosted(StatusBarNotification sbn) {
        // Notification is posted.
        outputNotificationData(sbn, "Notification Posted : ");
    }

    @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);
    }
}
```

上記の通り、NotificationListenerService を使い、ユーザーの許可を得ることで、Notification を読み取ることが可能になるが、Notification に含まれる情報には端末のプライベート情報が含まれることが多いため、取り扱いには十分な注意が必要である。

4.10.3.2 特定の Window をパススルーするタッチについて

Android 12 以降から、アプリが安全ではないオーバーレイ表示をしている場合、特定の Window をパススルーするタッチはブロックされる。この動作の変更は、targetSdkVersion に関係なく、Android 12 で実行されるすべてのアプリに適用される。

安全ではないオーバーレイ表示とは以下のようなものを指す。

- レイヤーに TYPE_APPLICATION_OVERLAY、フラグに FLAG_NOT_TOUCHABLE を使用する、SYSTEM_ALERT_WINDOW 権限を必要とするオーバーレイ表示
- FLAG_NOT_TOUCHABLE フラグを使用する Activity Window

ただし、以下の場合においては、パススルーするタッチは引き続き可能。

- アプリ内でのみオーバーレイ表示している場合
- レイヤーに TYPE_ACCESSIBILITY_OVERLAY、TYPE_INPUT_METHOD などが指定された信頼済みの Win-

dow である場合

- ルート View が GONE または INVISIBLE に設定された不可視の Window である場合
- alpha 値が 0 に設定された Window である場合
- レイヤーに TYPE_APPLICATION_OVERLAY を指定された Window のうち alpha 値が規定値以下である場合[*35]

アプリが安全ではないオーバーレイ表示をしている場合、ユースーケースに応じ次の API のいずれかを利用することが推奨される。

Bubble

Bubble は、Notification を使いやすくするため Android 11 から追加された機能で、メッセージなどを受信した際に、メッセージの通知を「ふきだし」で他のアプリの上に表示させることで、受信したアプリに切り替えることなく表示や返信をすることができる。

Picture-in-Picture (PIP)

PIP は、ユーザーがアプリ間を移動したり、メイン画面でコンテンツを閲覧したりしているときであっても、画面の隅に固定された小さな Window にコンテンツを表示することができる機能である。Android 8.0 以降で利用できる。

Notification

Notification は、デバイスの使用を最小限に抑えながら、ユーザーにリマインダー、他の人からのメッセージ、または他のアプリからのタイムリーな情報を提供する標準的な方法である。ユーザーは通知をタップしてアプリを開くか、通知から直接アクションを実行できる。

Snackbar

Snackbar は、アプリ起動中に短時間メッセージを表示する通知機能。

Toast

Snackbar 同様、短時間メッセージを表示する通知機能。アプリがバックグラウンドにあるときにメッセージを表示する必要がある場合には Toast を使用する。

信頼できないタッチを許可するには、ターミナルウィンドウで次の adb コマンドを実行する。

```
# 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
```

信頼できないタッチをブロックするデフォルトの動作に戻すには、ターミナルウィンドウで次の adb コマンドを実行する。

[*35] 規定値とは InputManager.getMaximumObscuringOpacityForTouch() で得られる値を指す。複数 Window がオーバーラップしている場合は alpha 値の総和が規定値以下である必要がある。

```
# 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 通知に関する実行時の権限

Android 13 から、Notification の通知機能を利用するにはユーザーの事前許可が必要になった。この動作変更は、targetSdkVersion に関係なく Android 13 プラットフォームで動作する全てのアプリに適用される。実装方法は targetSdkVersion により異なっており、それぞれの場合における実装方法を以下に述べる。

targetSdkVersion 32 以下の場合

Notification の通知機能を利用するには、createNotificationChannel を使用し、ユーザーに権限の付与をリクエストする。createNotificationChannel 実行時に権限ダイアログが表示され、ユーザーは次のアクションを実行できるようになる。

- [許可] を選択する
- [許可しない] を選択する
- どちらのボタンも押さずにダイアログをスワイプして閉じる

フォアグラウンドサービスとして動作するアプリの実装例は以下の通り。

```
NotificationManager notificationManager = (NotificationManager)context.
↪getSystemService(Context.NOTIFICATION_SERVICE);

NotificationChannel channel = new NotificationChannel(defaultId, name , importance);

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);
```

上記のコードを Android 13 プラットフォーム上で実行した際の権限ダイアログ、およびユーザーが [許可] または [許可しない] を選択した場合の通知ドロワーの様子は以下の通り。

図 4.10.4　権限ダイアログ

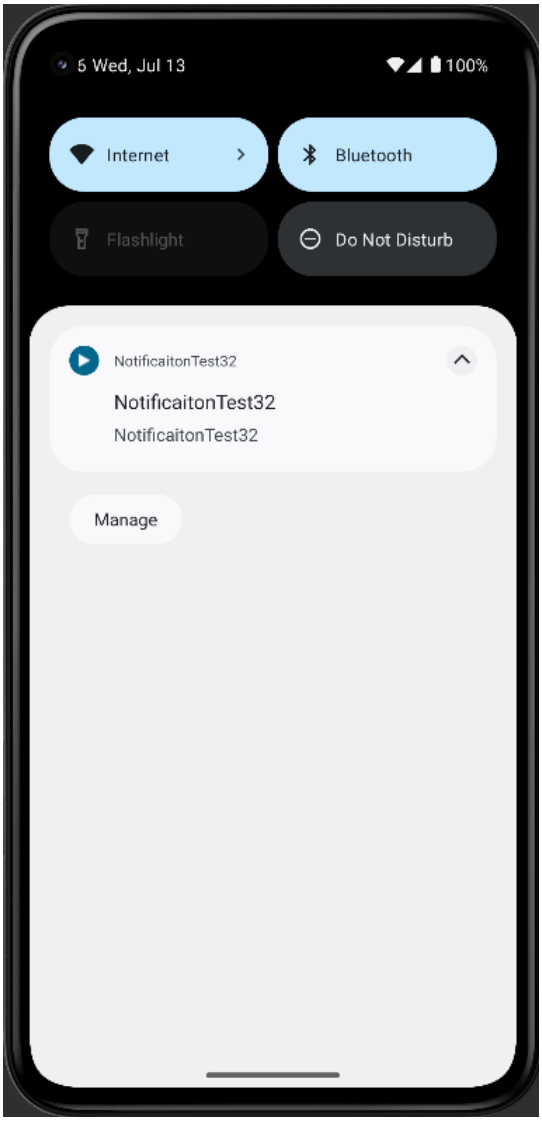

図 4.10.5　通知ドロワー [許可]

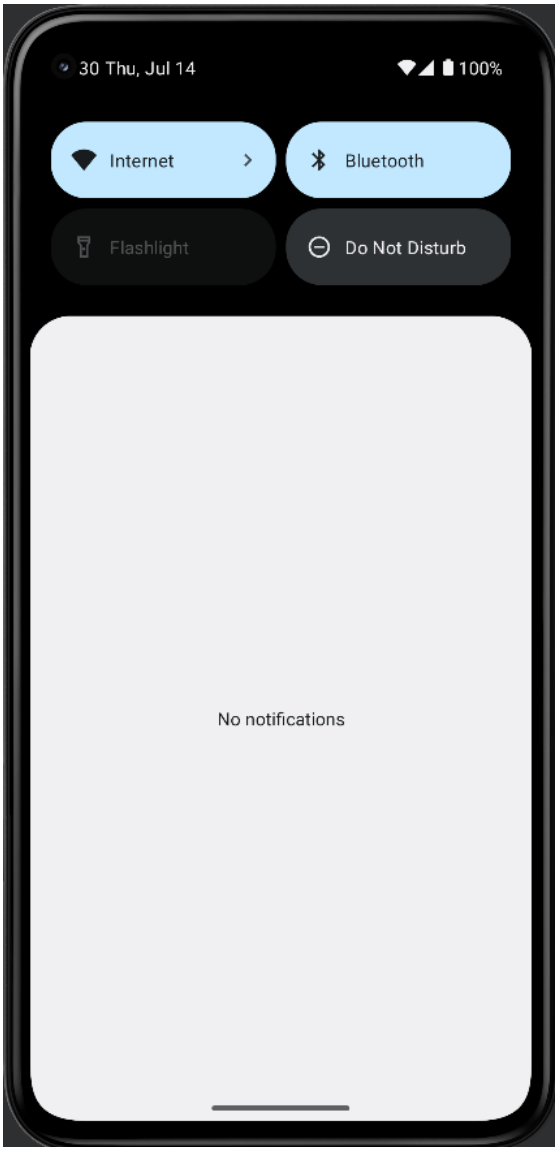

図 4.10.6　通知ドロワー [許可しない]

targetSdkVersion 33 以上の場合

上記のコードを、targetSdkVersion 33 以上に変更し、同じように Android 13 プラットフォーム上で実行した場合、権限ダイアログは表示されず、通知ドロワーにも Notification は表示されない。アプリ個別の設定から確認しても Notification 機能の ON/OFF 設定自体が無効化されている。

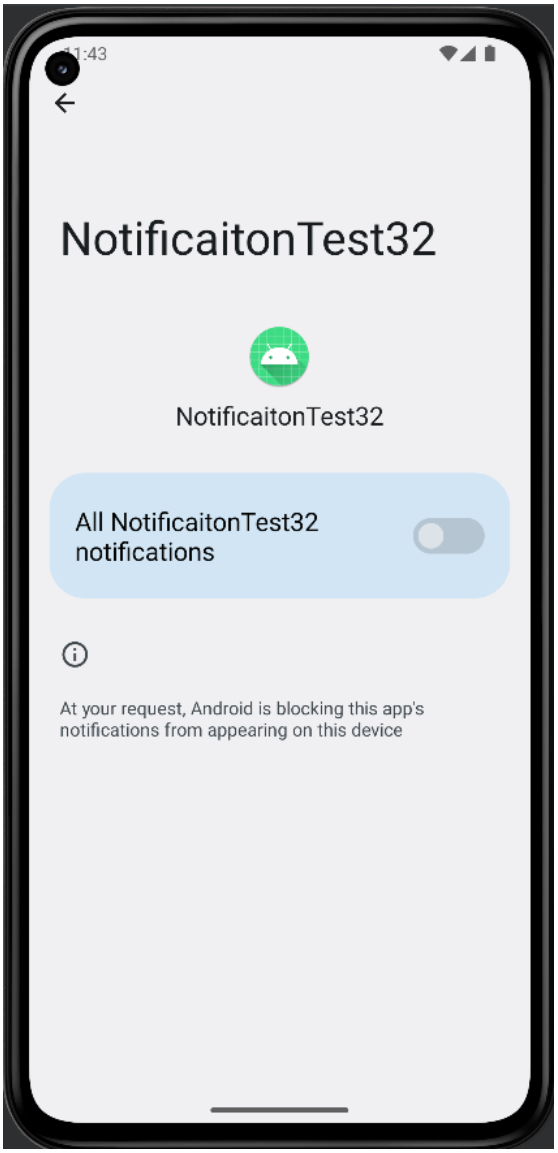

図 4.10.7　Notification 機能の ON/OFF 設定自体が無効化

これは、Android 13（API Level 33）から、通知を送信する新しい実行時の権限 POST_NOTIFICATIONS が導入されたためで、targetSdkVersion 33 以上でビルドする場合は、マニフェストファイルで POST_NOTIFICATIONS を宣言すること、そして別途、権限ダイアログを表示する実装が必要になる。

マニフェストファイルで宣言する例は以下の通り。

```
<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"/>
```

また、権限ダイアログを表示する実装例は以下の通り。

```
private ActivityResultLauncher<String> requestPermissionLauncher =
        registerForActivityResult(new ActivityResultContracts.RequestPermission(),␣
↪isGranted -> {
```

(次のページに続く)

(前のページからの続き)

```
            if (isGranted) {
                // Permission is granted. Continue the action or workflow in your
                // 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) {
    // 中略
    buttonStart.setOnClickListener( v -> {
        // 中略
        requestPermissionLauncher.launch(Manifest.permission.POST_NOTIFICATIONS);
        startForegroundService(intent);
    });
```

上記のコードを Android 13 プラットフォーム上で実行した際の権限ダイアログの様子は以下の通り。

図 4.10.8　権限ダイアログ

targetSdkVersion 33 以上とする場合のこれら変更への対応は「できるだけ早く」行うことが推奨されている。

以下に公式の文章を引用する[*36]。

> 「この機能による制御の強化と柔軟性の向上を活用するため、できるだけ早く Android 13 をターゲットに設定することを強くおすすめします。引き続き 12L（API Level 32）以下をターゲットとする場合は、アプリの機能のコンテキストに応じて権限をリクエストする機会を失うことになります。」

4.10.3.4 進行中を示す Notification の動作の変更

音楽メディアの再生中やダウンロード中など、処理が進行中であることを示す Notification は、FLAG_ONGOING_EVENT を設定することで、通知ドロワーに残り続けるよう設定可能となっている。設定は以下のいずれかの方法で行う。

- Notification.Builder#setOngoing(true)
- NotificationCompat.Builder#setOngoing(true)

Notification.Builder#setOngoing(true) での設定例

[*36] https://developer.android.com/about/versions/13/changes/notification-permission

```
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);
```

こうすることでユーザーが誤って Notification を閉じてしまうことを防ぐことができた。

しかし、Android 14 からこの動作が変更となり、FLAG_ONGOING_EVENT が設定されていても、ユーザーは Notification を閉じることが可能となっている。上記のコードを、Android 11 と Android 14 で実行した際の挙動の違いを以下に示す。

まずは Android 11 での挙動。

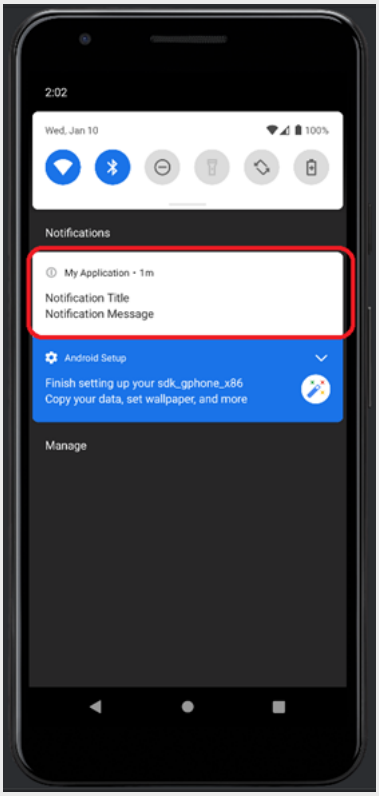

図 4.10.9　Android 11 の Notification

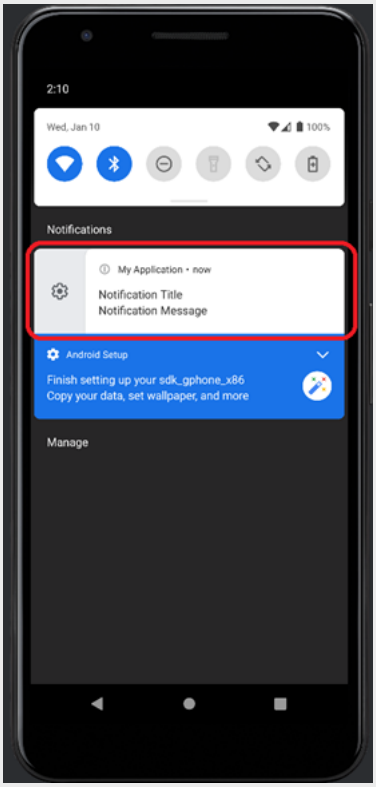

図 4.10.10　右にスワイプしても通知は消えない

次に Android 14 での挙動。

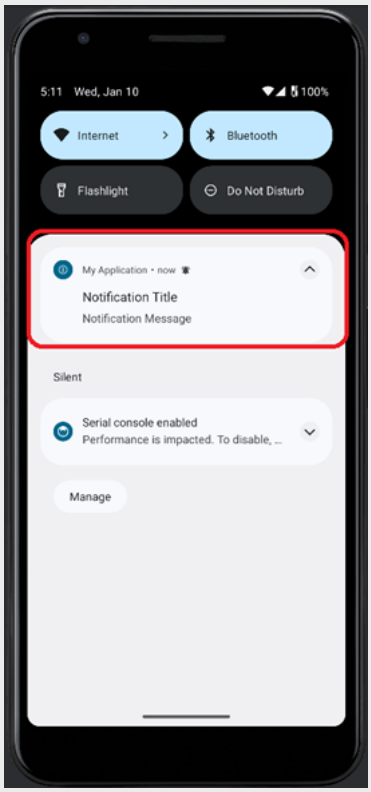

図 4.10.11　Android 14 の Notification

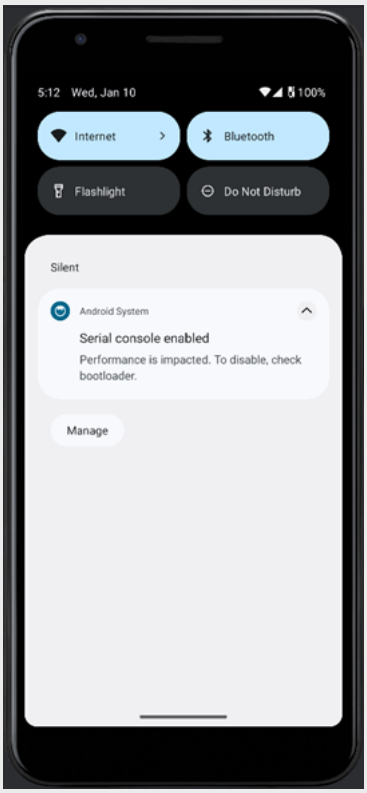

図 4.10.12　右にスワイプすることで通知は消える

ただし、上記の挙動は以下の状況では適用されず、Notification は引き続き通知ドロワーに残り続ける。

- スマートフォンがロックされている場合
- ユーザーが [すべて消去] 通知アクションを選択した場合（誤って消去しないようにするため）

また、以下のユースケースにおいても、上記の挙動は適用されない。

- CallStyle の通知
- エンタープライズ向けの Device Policy Controller（DPC）とサポートパッケージ
- メディア通知
- デフォルトの検索セレクタパッケージ

4.11 共有メモリを使用する

Android OS には以前から共有メモリ機構が存在しており android.os.MemoryFile として提供されていたが、複数のアプリで共有するための API やアクセス制御が直接的には提供されておらず、一般のアプリには使いにくいものであった。Android 8.1 (API Level 27) で android.os.SharedMemory パッケージが導入され、一般のアプリからも共有メモリ機構を比較的簡単に利用できるようになった。Android 8.1 の時点で MemoryFile は SharedMemory のラッパーとなり、SharedMemory の方を利用することが推奨されている。ここでは、この SharedMemory API を使用する際にセキュリティ上で注意すべき点を解説する。

後述するようにこの API は、あるアプリの Service が共有メモリを作成し、それらを他のアプリに提供することで提供されたアプリとメモリを共有する構成を前提とした作りになっている。したがって「4.4. *Service* を作る・利用する」で

解説されているすべての内容が、共有メモリを提供するアプリおよびそれを利用するアプリについても当てはまる。未読の方は先に「4.4. *Service* を作る・利用する」を読んでから、以下の説明に進むことを推奨する。

なお、SharedMemory API には対応する Android Support Library が無い。したがって SharedMemory を利用するアプリを API Level 27 未満の端末で動作させたい場合は、C 言語レベルの API を JNI でラップするなどして同等の仮想メモリ機構を実装し、バージョンによって使い分けるなどの措置が必要である。

4.11.1 Android 共有メモリの概要

共有メモリは複数のアプリの間で同一の物理メモリ領域を共有する機構である。

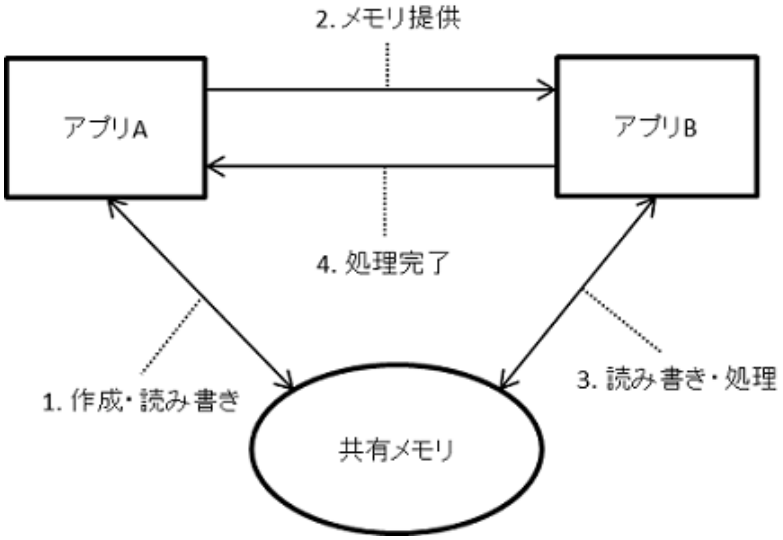

図 4.11.1　共有メモリの概要

上の図ではアプリ A とアプリ B で共有メモリを使用している様子を示している。アプリ A が共有メモリオブジェクトを作成し、それをアプリ B に提供している。アプリ A で共有メモリを提供する役割はアプリ A の Service が担う。アプリ B はこの Service に接続して共有メモリを要求・取得し、共有メモリで必要な処理が済んだら、利用が完了したことをアプリ A に知らせる。

共有メモリを利用することによって、例えば大きな画像のビットマップデータなど、通常のプロセス間通信で許されている最大サイズ (1MB[*37]) を超えるデータを複数のプロセスで共有することができる。また、デバイス全体で消費するメモリ量を削減でき、通常のメモリアクセスとなるため、非常に高速なプロセス間通信が実現できる。しかし、複数のアプリが同時並行的にアクセスすることになるため、データの整合性を維持するための考慮が必要となる場合がある。これを避けるためにはアプリ間での排他制御を行ったり、メモリ領域を適切に分割して互いのアクセスに干渉しないように考慮するなどの注意深い設計が必要である。

上述のように Android SDK の共有メモリ API は、Service が共有メモリオブジェクトを作成して他のプロセスへ供給する形態をとる。共有メモリのクラス (android.os.SharedMemory) は Parcelable として定義されているため、共有メモリのインスタンスを Binder 経由で他のプロセスに容易に渡すことができる。後で示すサンプルコードでの Service とクライアントとの間のやりとりの概要は（Service の作りによって大きく異なることもあるが)、次の図に示すような構造となる：

[*37] https://developer.android.com/guide/components/activities/parcelables-and-bundles

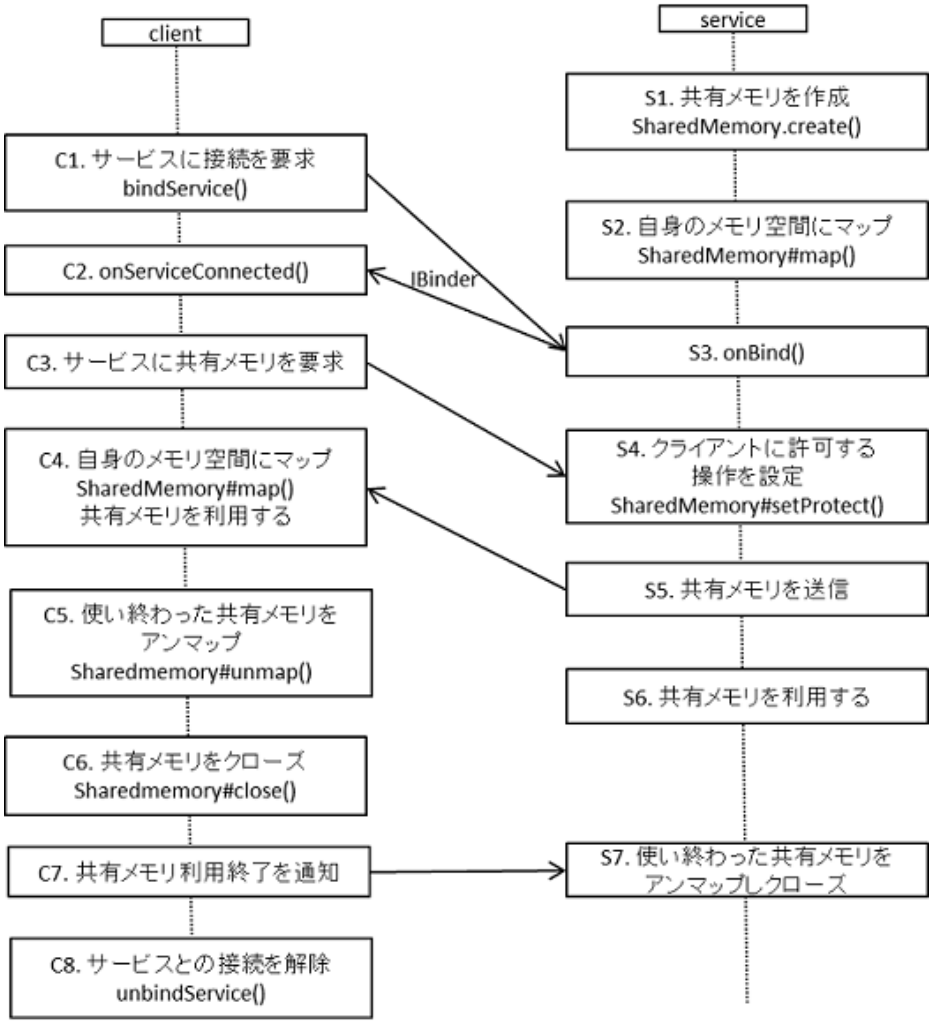

図 4.11.2　共有メモリ Service とクライアントのやりとり

- S1. Service は SharedMemory.create() により共有メモリを作成する
- S2. Service 自身が共有メモリを利用する場合 SharedMemory#map() によって自身のメモリ空間に共有メモリをマップする
- C1. クラアントは明示的 Intent を使用し Context#bindService() によってサービスに接続する
- S3. クライアントからの接続要求が届くと、Service の onBind() call back が呼び出される。Service はここで (あれば) 必要な前処理を行い、クライアントへ接続用の IBinder を返す。
- C2. Service が onBind() を実行した際の戻り値 (IBinder のインスタンス) は、クライアント側の onServiceConnected() callback の引数として返される。以降はこの IBinder を利用して Service との通信を行う。
- C3. クライアントは Service に対して共有メモリを要求する。
- S4. Service はクライアントからの共有メモリ要求を受け、クラアントが共有メモリをアクセスする際に許可する操作 (読み・書き) を設定する。
- S5. Service はクライアントへ共有メモリオブジェクトを渡す。
- C4. クライアントは受け取った共有メモリにアクセスするため、自身のアドレス空間へ共有メモリをマップし、利用する。
- C5, C6. クライアントが共有メモリ利用が終了したとき、自身のメモリ空間からアンマップし (C5)、共有メモリを

クローズ (C6) する。

- C7. クライアントはその後、共有メモリの利用が終了したことをサーバーへ通知する。
- C8. クライアントは Service との接続を解除する。
- S7. Service はクライアントから利用終了のメッセージを受け取ったのちに、自身も共有メモリをアンマップしクローズする。

上の C2. 項の onServiceConnected() は、android.content.ServiceConnection クラスを実装したクラスで定義する。具体例は後で示すサンプルコードを参照してほしい。IBinder を利用した通信方法にはいくつかのやり方があるが、サンプルコードでは Messenger を利用している。

4.11.2 サンプルコード

先に述べた通り、共有メモリを作成して他のアプリに提供する側は Service として実装する。そのため、機能や情報共有のセキュリティという観点からは「4.4. *Service* を作る・利用する」で述べられている事と本質的な違いはない。下図に 4.4. の分類に倣い、どのような相手とメモリを共有するかという視点で判定フローを示す。

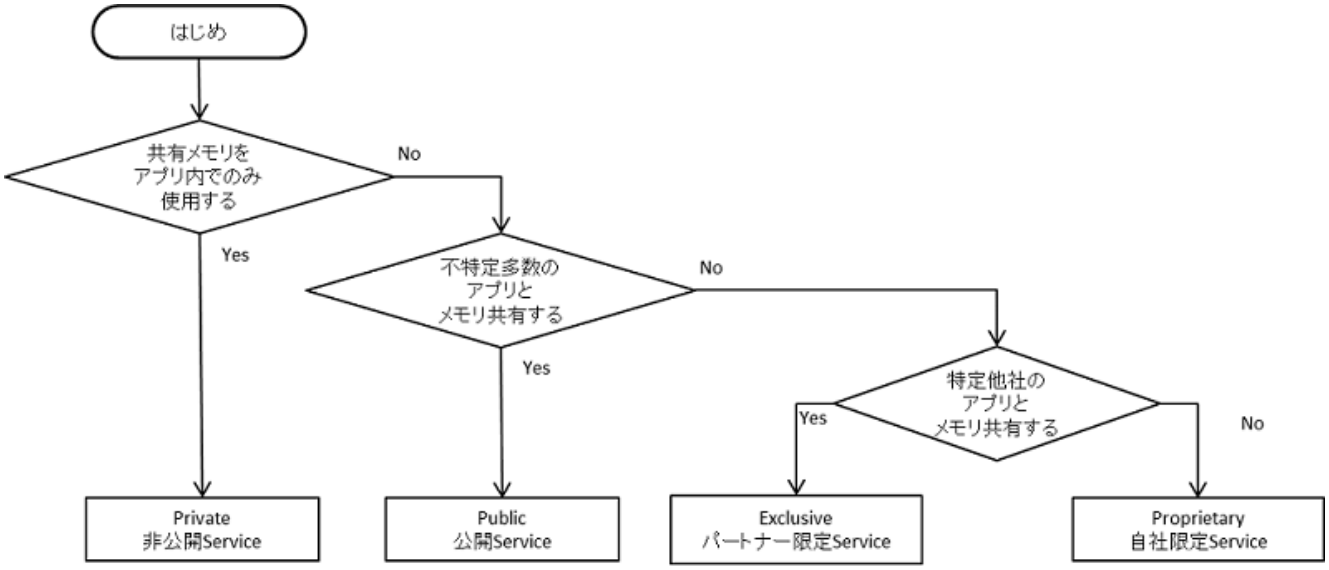

図 4.11.3　共有メモリ Service タイプの選択フロー

「4.4.1. サンプルコード」の 表 4.4.1 で、Service の実装方法について述べられているが、共有メモリの場合は、他のアプリとの共有は binder 経由で行う必要がある。そのため startService 型や IntentService 型の Service として実装することはできない。したがって下の表のようになる。

表 4.11.1　Service の実装方法（共有メモリの場合）

分類	非公開 **Service**	公開 **Service**	パートナー限定 **Service**	自社限定 **Service**
startService 型	-	-	-	-
IntentService 型	-	-	-	-
local bind 型	o	-	-	-
Messenger bind 型	o	o	-	**o***
AIDL bind 型	o	o	o	o

全体の作りについては、「4.4.1. サンプルコード」に掲載したものと本質的に同じである。また共有メモリに特有の事項については全てのケースで共通のため、具体的なサンプルコードは上の表で*印のついた、自社限定 Service に限定して示す。したがって他のケースで共有メモリを利用する場合は、「4.4.1.1. 非公開 *Service* を作る・利用する」から「4.4.1.3. パートナー限定 *Service*」にかけて記載されている内容を参照されたい。

4.11.2.1 非公開 Service を作る・利用する

この場合アプリに含まれる複数のプロセス間で非公開 Service が作成した共有メモリを共有する構造となる。またこの非公開 Service はアプリのメインプロセスとは独立したプロセスとして起動することとなる。

ポイント:

1. 共有メモリを作成する Service は exported=”false” により、明示的に非公開設定する
2. あるアプリ内プロセスが他のプロセスが書き込んだデータを参照する場合、同一アプリ内のプロセスであっても安全性を確認する
3. メモリを共有するのは同一アプリ内のプロセスであるから、センシティブな情報を共有しても良い

「4.4.1.1. 非公開 *Service* を作る・利用する」のサンプルコードでは Intent により Service を利用していたが、共有メモリの場合は Intent 経由ではメモリリソースを共有することができないため、local bind 型、Message bind 型、AIDL bind 型のいずれかの方法を用いる必要がある。

4.11.2.2 公開 Service を作る・利用する

「4.4.1.2. 公開 *Service* を作る・利用する」で述べた通り、公開 Service は不特定多数のアプリに利用されることを想定した Service である。そのためマルウェアが利用することも想定しなければならない。基本的に 4.4.1.2. のポイントで述べた通りの注意が必要であるが、以下ではこれを共有メモリの立場から言い換えてポイントを示す。

ポイント (Service を作る) :

1. exported=”true” により、明示的に公開設定する
2. サービスの起動やメモリ共有のためのリクエストなどに含まれるパラメータやデータの安全性を確認する
3. 共有メモリによってセンシティブな情報を共有しない

ポイント (Service を利用する): 1. センシティブな情報を共有メモリに書き込んではならない 2. 他のアプリが書き込んだデータを参照する場合は、その安全性を確認する

4.11.2.3 パートナー限定 Service

「4.4.1.3. パートナー限定 *Service*」で示されていることと本質的に同様であるが共有メモリの観点から言い換えて、以下にポイントを示す (4.4.1.3. のサンプルコードと同じく、AIDL bind 型の Service を想定する)。

ポイント (Service を作る) :

1. Intent Filter を定義せず、exported=”true” を明示的に宣言する
2. 利用元アプリの証明書がホワイトリストに登録されていることを確認する
3. onBind(onStartCommand, onHandleIntent) で呼び出し元がパートナーかどうか判別できない
4. パートナーアプリからの Intent であっても、受信 Intent の安全性を確認する

5. パートナーアプリに開示してよい情報に限り共有メモリに書き込んでも良い

ポイント (Service を使う):

1. 利用先のパートナー限定 Service アプリの証明書がホワイトリストに登録されていることを確認する
2. 利用先パートナー限定アプリに開示してよい情報のみ共有メモリに書き込んでも良い
3. 明示的 Intent によりパートナー限定 Service を呼び出す
4. パートナー限定アプリが書き込んだ情報であっても、データの安全性を確認する

4.11.2.4 自社限定 Service

ここでは、外部に公開された Service で共有メモリを提供するが、共有メモリを提供する先は自社製アプリに限定する例を示す。「4.4.1.4. 自社限定 *Service*」の例と同じく Messenger bind 型の Service となっている。背景にある考え方や設定については 4.4.1.4. に解説があるので、未読の方は 4.4.1.4. を先に参照されたい。

Service 側アプリのサンプルコード (Messenger bind 型)

以下にポイントを示すが、1 項から 5 項および 7 項は「4.4.1.4. 自社限定 *Service*」で提示されているものであり、共有メモリ特有のものは 6 項のみである。

ポイント:

1. 独自定義 Signature Permission を定義する
2. 独自定義 Signature Permission を要求宣言する
3. Intent Filter を定義せず、exported=”true” を明示的に宣言する
4. 独自定義 Signature Permission が自社アプリによって定義されていることを確認する
5. 自社アプリからの Intent であっても、受信 Intent の安全性を確認する
6. クライアントに共有メモリを渡す前に SharedMemory#setProtect() によってクライアント側が可能な操作を制限する
7. APK を Export するときに、利用元アプリと同じ開発者鍵で APK を署名する

この例では簡単のため共有メモリをアロケートする Service とそれを利用する Activity を同じアプリケーション内で定義している (Service は同じアプリ内の別プロセスとして起動されるようにしている)。そのため独自定義 Signature Permission の定義と利用宣言が両方ともに Manifest ファイルに記載されている。

AndroidManifest.xml

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

  <!-- ★ポイント 1★ 独自定義 Signature Permission を定義する -->
  <permission android:name="org.jssec.android.sharedmemory.inhouseservice.messenger.MY_
↪PERMISSION"
              android:protectionLevel="signature" />
```

(次のページに続く)

(前のページからの続き)

```
  <!-- ★ポイント 8★ 独自定義 Signature Permission を利用宣言する -->
  <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>
    </activity>
    <!-- Messenger を利用した Service -->
    <!-- ★ポイント 2★ 独自定義 Signature Permission を要求宣言する -->
    <!-- ★ポイント 3★ Intent Filter を定義せず exported="true"を明示的に設定する -->
    <!-- 簡単のため共有メモリを提供するサービスを同じアプリの別プロセスとする -->
    <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
 *
```

(次のページに続く)

(前のページからの続き)

```
 * 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;

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 {
    // 自社の Signature Permission
    private static final String MY_PERMISSION =
        "org.jssec.android.sharedmemory.inhouseservice.messenger.MY_PERMISSION";

    // 自社の証明書のハッシュ値
    private static String sMyCertHash = null;
    private static String myCertHash(Context context) {
        if (sMyCertHash == null) {
            if (Utils.isDebuggable(context)) {
                // debug.keystore の"androiddebugkey"の証明書ハッシュ値
                sMyCertHash = "0EFB7236 328348A9 89718BAD DF57F544 D5CCB4AE B9DB34BC␣
```

(次のページに続く)

(前のページからの続き)

```
↪1E29DD26 F77C8255";
            } else {
                // keystore の"my company key"の証明書ハッシュ値
                sMyCertHash = "D397D343 A5CBC10F 4EDDEB7C A10062DE 5690984F 1FB9E88B␣
↪D7B3A7C2 42E142CA";
            }
        }
        return sMyCertHash;
    }

    private final String TAG = "SHM";

    // クライアントへ送信するメッセージ文字列
    private final String greeting = "Hi! I send you my memory. Let's Share it!";
    // ページサイズは 4K bytes
    public static final int PAGE_SIZE = 1024 * 4;
    // 共有メモリを 2 つ使用する。クライアントは下の識別子を指定してどの共有メモリが欲しいかを指定する
    public static final int SHMEM1 = 0;
    public static final int SHMEM2 = 1;

    // 共有メモリのインスタンス
    // mSHMem1: クライアントへ渡すデータ用として使用する
    private SharedMemory mSHMem1 = null;
    // mSHMem をマップするためのバイトバッファ
    private ByteBuffer m1Buffer1;
    // mSHMem2: クライアントからのデータ受け取り用として使用する
    private SharedMemory mSHMem2 = null;
    // mSHMem2 をマップするためのバイトバッファ。二つの領域に分割して用いる
    private ByteBuffer m2Buffer1;
    private ByteBuffer m2Buffer2;
    // すべてのバイトバッファが正常にマップされたときに true となる
    private boolean mBufferMapped = false;


    // この例ではクライアントとは Messenger 機構を利用して通信する
    // 以下はそこで使用するメッセージ識別子
    public static final int MSG_INVALID = Integer.MIN_VALUE;
    public static final int MSG_ATTACH  = MSG_INVALID + 1; // クライアントが SHMEM1 アロケート
要求
    public static final int MSG_ATTACH2 = MSG_ATTACH + 1;  // クライアントが SHMEM2 アロケート
要求
    public static final int MSG_DETACH  = MSG_ATTACH2 + 1; // クライアントが SHMEM1 を使い終
わった
    public static final int MSG_DETACH2 = MSG_DETACH + 1;  // クライアントが SHMEM2 を使い終
わった
```

(次のページに続く)

(前のページからの続き)

```
    public static final int MSG_REPLY1  = MSG_DETACH2 + 1; // クライアントからの返答 1
    public static final int MSG_REPLY2  = MSG_REPLY1 + 1;  // クライアントからの返答 2
    public static final int MSG_END     = MSG_REPLY2 + 1;  // Service が終了を宣言

    // クライアントから受け取った Message を処理する Handler
    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();

    // クライアントからのデータを受信するときに利用する Messenger
    private final Messenger mMessenger = new Messenger(mHandler);

    // バインド時に Messenger から Binder を取得しクライアントへ渡す
    @Override
    public IBinder onBind(Intent intent) {
        // ★ポイント 4★ 独自定義 Signature Permission が自社アプリにより定義されていることを確認する
        if (!SigPerm.test(this, MY_PERMISSION, myCertHash(this))) {
```

(次のページに続く)

(前のページからの続き)

```
            Toast.makeText(this,
                           "独自定義 Signature Permission が自社アプリによって定義されていない。",
                           Toast.LENGTH_LONG).show();
            return null;
        }

        // ★ポイント 5★ 自社アプリからの Intent であっても、受信 Intent の安全性を確認する
        // サンプルにつき割愛。「3.2 入力データの安全性を確認する」を参照。
        String param = intent.getStringExtra("PARAM");
        // Toast.makeText(this, String.format("パラメータ「%s」を受け取った。", param),
        //                                    Toast.LENGTH_LONG).show();
        Log.d(TAG, String.format("パラメータ [%s] を受け取った！ ", param));
        return mMessenger.getBinder();
    }

    // マップ構造
    // オフセットはページ境界でなければならない
    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;

    // 2 つの共有メモリをアロケートする
    private boolean allocateSharedMemory() {
        try {
            // クライアントへ渡すデータ用共有メモリ
            mSHMem1 = SharedMemory.create("SHM", PAGE_SIZE);
            // クライアントからのデータ受け取り用共有メモリ
            mSHMem2 = SharedMemory.create("SHM2", PAGE_SIZE * 2);
        } catch (ErrnoException e) {
            Log.e(TAG, "failed to allocate shared memory" + e.getMessage());
            return false;
        }
        return true;
    }

    // 指定の共有メモリをマップする
    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,
```

(次のページに続く)

(前のページからの続き)

```
                "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;
    }

    // 自分用に二つの共有メモリをマップ
    private void mapMemory() {
        // mSHMem1 : 読み書きする
        m1Buffer1 =
            mapShared(mSHMem1,
                 PROT_READ | PROT_WRITE | PROT_EXEC, SHMEM1_BUF1_OFFSET, SHMEM1_BUF1_
↪LENGTH);
        // mSHMem2 : 二つの領域に分割し、それぞれを読み書きする
        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;
    }

    // 共有メモリを解放する
    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();
```

(次のページに続く)

(前のページからの続き)

```
                mSHMem2 = null;
            }
            mBufferMapped = false;
        }
    }

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

        // インスタンス生成のタイミングで共有メモリをアロケート
        // アロケートできたら共有メモリをマップする
        if (allocateSharedMemory()) {
            mapMemory();
        }
    }

    // SHMEM1 をクライアントへ提供
    private void shareWith1(Message msg){
        // アロケートされていなかったらなにもしない
        if (!mBufferMapped) return;

        // ★ポイント 6★ クライアントに共有メモリを渡す前に SharedMemory#setProtect() によってクライアント側が可能な操作を制限する
        // mSHMem1 はクライアントに読み込みのみを許可する
        mSHMem1.setProtect(PROT_READ);

        // 上の setProtect(PROT_READ) 実行以前にマッピングが完了しているため、書き込みが可能
        // mBuffer の先頭にメッセージのサイズを入れ、続いてメッセージ文字列を追加
        m1Buffer1.putInt(greeting.length());
        m1Buffer1.put(greeting.getBytes());

        try {
            // 要求元に共有メモリオブジェクトを渡す
            Message sMsg = Message.obtain(null, SHMEM1, mSHMem1);
            msg.replyTo.send(sMsg);
        } catch (RemoteException e) {
            Log.e(TAG, "Failed to share" + e.getMessage());
        }
    }

    // SHMEM2 をクライアントへ提供
    private void shareWith2(Message msg) {
        if (!mBufferMapped) return;
```

(次のページに続く)

(前のページからの続き)

```
        // ★ポイント 6★ クライアントに共有メモリを渡す前に SharedMemory#setProtect() によってクライアント側が可能な操作を制限する
        // mSHMem2 はクライアントに書き込みを許可する
        mSHMem2.setProtect(PROT_WRITE);
        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 {
            // 要求元に共有メモリオブジェクトを渡す
            Message sMsg = Message.obtain(null, SHMEM2, mSHMem2);
            msg.replyTo.send(sMsg);
        } catch (RemoteException e){
            Log.e(TAG, "failed to share mSHMem2" + e.getMessage());
        }
    }

    // メモリの共有を止める
    private void unShare(Message msg){
        deAllocateSharedMemory();
    }

    // 不正なメッセージを受け取った
    private void invalidMsg(Message msg){
        Log.e(TAG, "Got an Invalid message: " + msg.what);
    }

    // クライアントから受け取った情報を取り出す
    // 先頭にサイズ、それに続いて文字バイト列が格納されているものとする
    private String extractReply (ByteBuffer buf){
        int len = buf.getInt();
        byte [] bytes = new byte[len];
        buf.get(bytes);
        return new String(bytes);
    }

    // この例では 2 種類のメッセージをクライアントから受け取る
    // こちらは m1Buffer1 にクライアントからの情報が格納されていることを想定
    private void gotReply(Message msg) {
        m1Buffer1.rewind();
        String message = extractReply(m1Buffer1);
        if (!message.equals(greeting)){
            Log.e(TAG, "my message was overwritten: " + message);
```

(次のページに続く)

(前のページからの続き)

```
        }
    }
    // こちらは m2Buffer1 にクライアントからの情報が格納されていることを想定
    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);
        // このリプライメッセージが来た場合は、共有メモリの利用を終了することをクライアントと取り決めている
        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;

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

(次のページに続く)

(前のページからの続き)

```
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{
            // sigPermName を定義したアプリのパッケージ名を取得する
            PackageManager pm = ctx.getPackageManager();
            PermissionInfo pi =
                pm.getPermissionInfo(sigPermName, PackageManager.GET_META_DATA);
            String pkgname = pi.packageName;
            // 非 Signature Permission の場合は失敗扱い
            if (pi.protectionLevel != PermissionInfo.PROTECTION_SIGNATURE) return false;
            // pkgname の実際のハッシュ値と正解のハッシュ値を比較する
            if (Build.VERSION.SDK_INT >= 28) {
                // ★ API Level >= 28 では Package Manager の API で直接検証が可能
                return pm.hasSigningCertificate(pkgname,
                                                Utils.hex2Bytes(correctHash),
                                                CERT_INPUT_SHA256);
            } else {
                // API Level < 28 の場合は PkgCert を利用し、ハッシュ値を取得して比較する
                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
```

(次のページに続く)

(前のページからの続き)

```
 * 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);
            if (pkginfo.signatures.length != 1) return null;  // 複数署名は扱わない
            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) {
```

(次のページに続く)

(前のページからの続き)

```
            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();
    }
}
```

★ポイント 7 ★ APK を Export するときに、利用元アプリと同じ開発者鍵で APK を署名する。

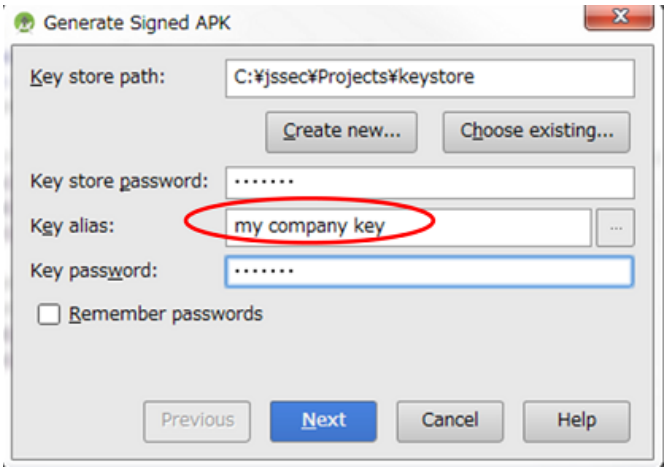

図 4.11.4　利用元アプリと同じ開発者鍵で APK を署名する

クライアント側のサンプルコード

ポイント: 8. 独自定義 Signature Permission を利用宣言する 9. 独自定義 Signature Permission が自社アプリにより定義されていることを確認する 10. 利用先アプリの証明書が自社の証明書であることを確認する 11. 利用先アプリは自社アプリであるから、センシティブな情報を送信してもよい 12. 明示的 Intent により自社限定 Service を呼び出す 13. APK を Export するときに、利用先アプリと同じ開発者鍵で APK を署名する

ここで示されているポイントは、すべて「4.4.1.4. 自社限定 *Service*」のクライアント側のポイントと同じであり、共有メモリ特有のものはない。共有メモリの利用については、下記のサンプルコードに基本的なものが示されているのでこれを参照してほしい。

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

  <!-- ★ポイント 1★ 独自定義 Signature Permission を定義する -->
```

(次のページに続く)

(前のページからの続き)

```
  <permission android:name="org.jssec.android.sharedmemory.inhouseservice.messenger.MY_
↪PERMISSION"
              android:protectionLevel="signature" />

  <!-- ★ポイント 8★ 独自定義 Signature Permission を利用宣言する -->
  <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>
    </activity>
    <!-- Messenger を利用した Service -->
    <!-- ★ポイント 2★ 独自定義 Signature Permission を要求宣言する -->
    <!-- ★ポイント 3★ Intent Filter を定義せず exported="true"を明示的に設定する -->
    <!-- 簡単のため共有メモリを提供するサービスを同じアプリの別プロセスとする -->
    <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.
```

(次のページに続く)

(前のページからの続き)

```
 * 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;
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";

    // Service へデータを送信するときに使用する Messenger
```

(次のページに続く)

(前のページからの続き)

```
    private Messenger mServiceMessenger = null;

    // 共有メモリオブジェクト
    private SharedMemory myShared1;
    private SharedMemory myShared2;
    // 共有メモリをマップする ByteBuffer
    private ByteBuffer mBuf1;
    private ByteBuffer mBuf2;

    // 利用先の Activity 情報
    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";

    // 自社の Signature Permission
    private static final String MY_PERMISSION =
        "org.jssec.android.sharedmemory.inhouseservice.messenger.MY_PERMISSION";

    // 自社の証明書のハッシュ値
    private static String sMyCertHash = null;
    private static String myCertHash(Context context) {
        if (sMyCertHash == null) {
            if (Utils.isDebuggable(context)) {
                // debug.keystore の"androiddebugkey"の証明書ハッシュ値
                sMyCertHash = "0EFB7236 328348A9 89718BAD DF57F544 D5CCB4AE B9DB34BC␣
↪1E29DD26 F77C8255";
            } else {
                // keystore の"my company key"の証明書ハッシュ値
                sMyCertHash = "D397D343 A5CBC10F 4EDDEB7C A10062DE 5690984F 1FB9E88B␣
↪D7B3A7C2 42E142CA";
            }
        }
        return sMyCertHash;
    }

    // Service と接続された状態で true となる
    private boolean mIsBound = false;

    // Service から受け取った Message を処理する Handler
    private class MyHandler extends Handler {
        @Override
        public void handleMessage(Message msg) {
            switch (msg.what) {
            case SHMService.SHMEM1:
```

(次のページに続く)

(前のページからの続き)

```
                // サービスから SHMEM 1 が提供された
                Log.d(TAG, "got SHMEM1");
                // Message.obj に共有メモリが格納されている
                myShared1 = (SharedMemory) msg.obj;
                useSHMEM1();
                break;
            case SHMService.SHMEM2:
                // サービスから SHMEM 2 が提供された
                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();

    // Service からデータを受信するときに使用する Messenger
    private Messenger mLocalMessenger = new Messenger(mHandler);

    // Service と接続する時に利用するコネクション。bindService で実装する場合は必要になる
    private class MyServiceConnection implements ServiceConnection {
        // Service に接続された場合に呼ばれる
        public void onServiceConnected(ComponentName className, IBinder service){
            mServiceMessenger = new Messenger(service);
            // 共有メモリ Service にバインドしたタイミングで一つ目の共有メモリを要求する
            sendMessageToService(SHMService.MSG_ATTACH);
        }
        // Service が異常終了して、コネクションが切断された場合に呼ばれる
        public void onServiceDisconnected(ComponentName className){
            mIsBound = false;
            mServiceMessenger = null;
        }
    }
    private MyServiceConnection mServiceConnection;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
```

(次のページに続く)

(前のページからの続き)

```
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        doBindService ();
    }

    // 共有メモリ Service に接続する
    private void doBindService () {
        mServiceConnection = new MyServiceConnection();
        if (!mIsBound) {
            // ★ポイント 9★ 独自定義 Signature Permission が自社アプリにより定義されていることを確
認する
            if (!SigPerm.test(this, MY_PERMISSION, myCertHash(this))) {
                Toast.makeText(this,
                               "独自定義 Signature Permission が自社アプリにより定義されていない。",
                               Toast.LENGTH_LONG).show();
                return;
            }
            // ★ポイント 10★ 利用先アプリの証明書が自社の証明書であることを確認する
            if (!PkgCert.test(this, SHM_PACKAGE, myCertHash(this))) {
                Toast.makeText(this,
                               "利用先サービスは自社アプリではない。",
                               Toast.LENGTH_LONG).show();
                return;
            }
        }
        Intent it = new Intent();
        // ★ポイント 11★ 利用先アプリは自社アプリであるから、センシティブな情報を送信してもよい
        it.putExtra("PARAM", "センシティブな情報");

        // ★ポイント 12★ 明示的インテントを用いて共有メモリサービスに bind する
        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;
    }

    // サーバーとの接続を切断する
    private void releaseService () {
        unbindService(mServiceConnection);
    }
```

(次のページに続く)

(前のページからの続き)

```
    // SHMEM1 を利用する例
    private void useSHMEM1 () {
        // SHMEM1 は読み込みのみが許可されているので PROT_READ 以外でマップすると例外が発生する
        // 例外は mapMemory() でハンドリングされる
        mBuf1 = mapMemory(myShared1, PROT_WRITE,
                          SHMService.SHMEM1_BUF1_OFFSET, SHMService.SHMEM1_BUF1_LENGTH);
        // PROT_READ でマップ
        mBuf1 = mapMemory(myShared1, PROT_READ,
                          SHMService.SHMEM1_BUF1_OFFSET, SHMService.SHMEM1_BUF1_LENGTH);
        // サービス側がセットしたデータを読み込む
        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();
        // read only のため、書き込むと ReadOnlyBufferException 例外が発生する
        try {
            mBuf1.putInt(0);
        } catch (ReadOnlyBufferException e){
            Log.e(TAG, "Write to read only buffer: " + e.getMessage());
        }
        // サービスへリプライ
        sendMessageToService(SHMService.MSG_REPLY1);
        // 続けて書き込み可能な共有メモリをリクエストする
        sendMessageToService(SHMService.MSG_ATTACH2);
    }

    // SHMEM2 を利用する例
    private void useSHMEM2 () {
        // SHMEM2 は write が許可されているため PROT_WRITE でマッピングできる

        // サービス側では PROT_WRITE を設定しているため PROT_READ | PROT_WRITE でマップすると例外が
発生する
        mBuf2 = mapMemory(myShared2, PROT_WRITE,
                          SHMService.SHMEM2_BUF1_OFFSET, SHMService.SHMEM2_BUF1_LENGTH);
        if (mBuf2 != null) {
            // しかし PROT_WRITE のみでも read はできる
            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);

            // マップした領域を超えてアクセスすると IndexOutOfBoundsException 例外が発生する
```

(次のページに続く)

(前のページからの続き)

```
            try {
                mBuf2.get(SHMService.SHMEM2_BUF1_LENGTH + 1);
            } catch (IndexOutOfBoundsException e){
                Log.e(TAG, "out of bound: " + e.getMessage());
            }

            // サービス側で設定したデータを上書きする
            String replyStr = "OK Thanks!";
            mBuf2.putInt(replyStr.length());
            mBuf2.put(replyStr.getBytes());
            // サービスへリプライ
            sendMessageToService(SHMService.MSG_REPLY2);
        }

        if (mBuf2 != null) SharedMemory.unmap(mBuf2);
        // mBuf とオーバーラップする領域を今度は PROT_WRITE | PROT_EXEC でマップ
        // サービス側で共有メモリの PROT_EXEC ビットをオフにしるため、これは失敗する
        // 発生する例外は mapMemory() でハンドリングされている
        mBuf2 = mapMemory(myShared2,PROT_WRITE | PROT_EXEC, 0, 64);

        // マップできる領域サイズはサービス側とは独立している。共有メモリのサイズを超えない限り任意のサイズをマップできる
        mBuf2 = mapMemory(myShared2, PROT_READ | PROT_WRITE,
                          SHMService.SHMEM2_BUF2_OFFSET, SHMService.PAGE_SIZE);
        if (mBuf2 != null) {
            int size = mBuf2.getInt();
            byte [] bytes = new byte[size];
            mBuf2.get(bytes);
            String msg = new String(bytes);
            Log.d(TAG, "Got another message from service: " + msg);
        }
    }
    // 指定の共有メモリをマップする
    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;
    }
```

(次のページに続く)

(前のページからの続き)

```
    // サービスへメッセージを送る
    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());
        }
    }

    // 使い終わった共有メモリを後始末する
    private void alloverNow() {
        // サービスへ使い終わったことを通知
        sendMessageToService(SHMService.MSG_DETACH);
        sendMessageToService(SHMService.MSG_DETACH2);
        // マップされた ByteBuffer を unmap
        if (mBuf1 != null) SharedMemory.unmap(mBuf1);
        if (mBuf2 != null) SharedMemory.unmap(mBuf2);
        // 共有メモリをクローズ
        myShared1.close();
        myShared2.close();
        mBuf1 = null;
        mBuf2 = null;
        myShared1 = null;
        myShared2 = null;
        // サーバーとのコネクションを切る
        releaseService();
    }
}
```

★ポイント 13 ★ APK を Export するときに、利用先アプリと同じ開発者鍵で APK を署名する。

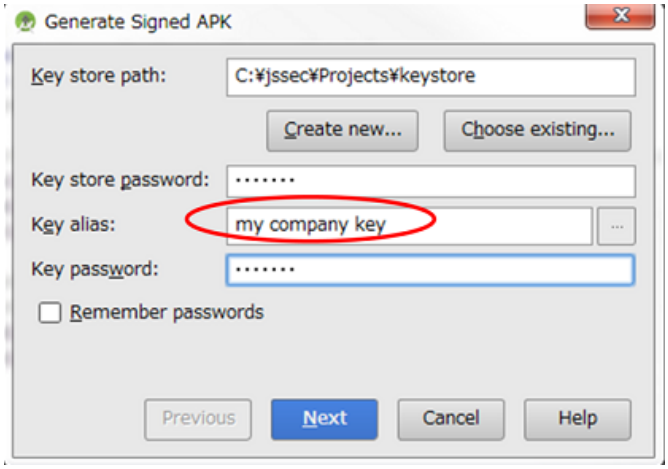

図 4.11.5　利用先アプリと同じ開発者鍵で APK を署名する

4.11.3 ルールブック

SharedMemory を利用する際には Service についてのルールブック (4.4.2. ルールブック) に記載のルールを守ること。それらに加えて、以下のルールを守ること。

1. 共有メモリを提供する側は利用側へ許可するアクセス権を適切に設定する *(必須)*
2. 共有メモリ上のデータはすべて共有するアプリから読まれることを前提に設計する *(必須)*

4.11.3.1 共有メモリを提供する側は利用側へ許可するアクセス権を適切に設定する (必須)

メモリを共有する場合、各アプリがメモリに対して可能な操作を設計上必要最低限の操作に制限し、情報の漏洩や改ざん・破壊を防止すべきである。SharedMemory オブジェクトを作成する Service は他のアプリと共有する前に SharedMemory#setProtect() を使用して共有メモリ全体に対して可能な操作を制限できる。SharedMemory オブジェクトに対して可能な操作の初期値は、読み・書き・実行である。このうち実行可能なメモリ領域は特別な理由がない限り、不正なコードの実行を未然に防ぐため使用すべきではない[*38]。また、他のアプリが共有メモリに書き込む必要のある場合は、別途専用の共有メモリを作成して提供することでより安全にメモリを共有することができる。

shMem.setProtect(PROT_READ | PROT_WRITE)

クライアント側でこの共有メモリ内の (一部または全体の) 領域をアクセスできるようにするには SharedMemory#map() を事前に実行する必要がある。その際引数の一つとしてメモリに対して行う操作を指定するが、事前に Service 側が SharedMemory#setProtect() で許可した以上の操作を指定することはできない。例えば Service 側が読み込みのみを許可しているにもかかわらず、クライアント側で書き込み操作を指定することはできない。下の例は Service から提供された SharedMemory オブジェクト ashMem を map() している例である。

```
ByteBuffer mbuf;
// Service が読み込みのみを許可している場合、下のコードは例外を引き起こす
mbuf = ashMem.map(offset, length, PROT_WRITE);
```

[*38] デバイス (使用している CPU アーキテクチャ) によっては、あるメモリ領域が読み込み可能であった場合、自動的に実行可能となるものがある。しかしその場合でも、その領域に対して書き込みを禁止することにより、他のアプリがその領域に実行可能なコードを書き込むのを防止することが可能である。

SharedMemory#setProtect() の引数は読み・書き・実行のそれぞれに対応したビットフラグ (PROT_READ、PROT_WRITE、PROT_EXEC) の論理的 OR である。下は SharedMemory オブジェクト shMem に対して読み込みと書き込みのみを許可している例である。

クライアント側でも setProtect() を呼び出して共有メモリ全体に許可される操作を設定し直すことは可能だが、map() と同様に先に Service によって許可された操作以上の操作を許可するように設定することはできない。

4.11.3.2 共有メモリ上のデータはすべて共有するアプリから読まれることを前提に設計する (必須)

上述のように他のアプリとメモリを共有する場合、提供する側で事前に共有メモリに対するアクセス権 (読み・書き・実行) を設定できる。しかしフラグを PROT_WRITE のみにして書き込みだけを許可しても、メモリの読み込みを禁止することができない場合がある。言い換えると、デバイスが使用しているメモリ管理ユニット (MMU) が書き込みのみを許すメモリアクセスをサポートしていない場合は、あるメモリ領域が書き込み可能であると読み込みも可能となるということである。実際にこのようなデバイスは多いと考えられ、そのため共有メモリの内容は他のアプリに知られることを前提とした設計をしなければならない。

```
// Service 側で SharedMemory#setProtect(PROT_WRITE) で書き込みのみを許可しているものとする
// クライアント側で map により PROT_WRITE でマップしても、読み込みが可能である場合が多い
ByteBuffer buf;
buf = ashMem.map(offset, length, PROT_WRITE);
// その場合読み込み操作はエラーとならない
int len = buf.getInt();
byte [] bytes = new byte[len];
buf.get(bytes);
```

フラグに PROT_NONE を指定し、いかなる操作もできないようにすることは可能であるが、その時点で共有の意味は失われる。

4.11.4 アドバンスト

4.11.4.1 共有メモリの実際

これまでの説明では共有メモリを複数のアプリ間でメモリを共有する機構として述べてきた。しかし、実際のところ共有メモリは同一の物理メモリ領域を複数のプロセス間で共有する機構である。各々のプロセスは共有する物理メモリ領域を自身のアドレス空間にマップしてアクセスする (SharedMemory#map() はこれを行なっている)。Android の共有メモリの場合、マップされたメモリ領域は (Java 言語の場合) 一つの ByteBuffer オブジェクトである (ページサイズを超えた共有メモリをアロケートした場合、共有される物理メモリ領域は連続した領域ではなく複数の非連続なページに分割されているのが一般的であるが、プロセスのアドレス空間上にマップした場合は連続したアドレス空間となる。)

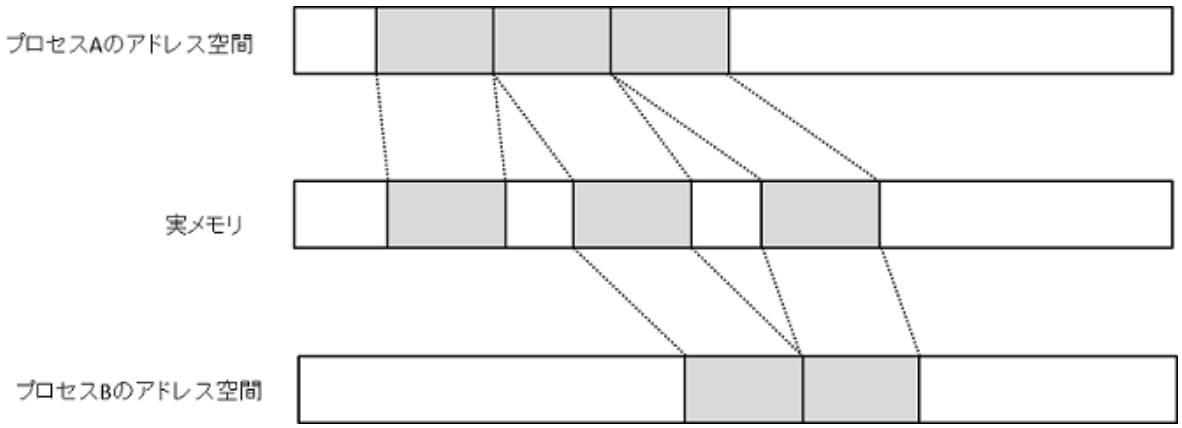

図 4.11.6　実メモリとプロセスのアドレス空間

Android OS を含む Unix 系の OS では、接続されているターミナルや USB デバイスなどの周辺機器をデバイスファイルという概念で抽象化し、仮想的なファイルとして取り扱う。Android OS の共有メモリもその例外ではなく、/dev/ashmem というデバイスファイルがこれに相当する。このデバイスファイルをオープンすると、通常のファイルをオープンした場合と同様にファイルディスクリプタが返され、これを通じて共有メモリへのアクセスが行われる。このファイルディスクリプタは通常のファイルの場合とおなじく mmap() によってプロセスのアドレス空間にマップすることができる。mmap() は Unix 系 OS の標準的なシステムコールであり、様々なデバイスに対応するデバイスファイルのファイルディスクリプタを取り、それを呼び出しプロセスのアドレス空間にマップする機能を提供している。Android OS の共有メモリでもこれを利用している。マップされたアドレス空間はプログラムからはバイト列として見える (Java の場合は上述のように ByteBuffer、C 言語レベルでは char * である)。

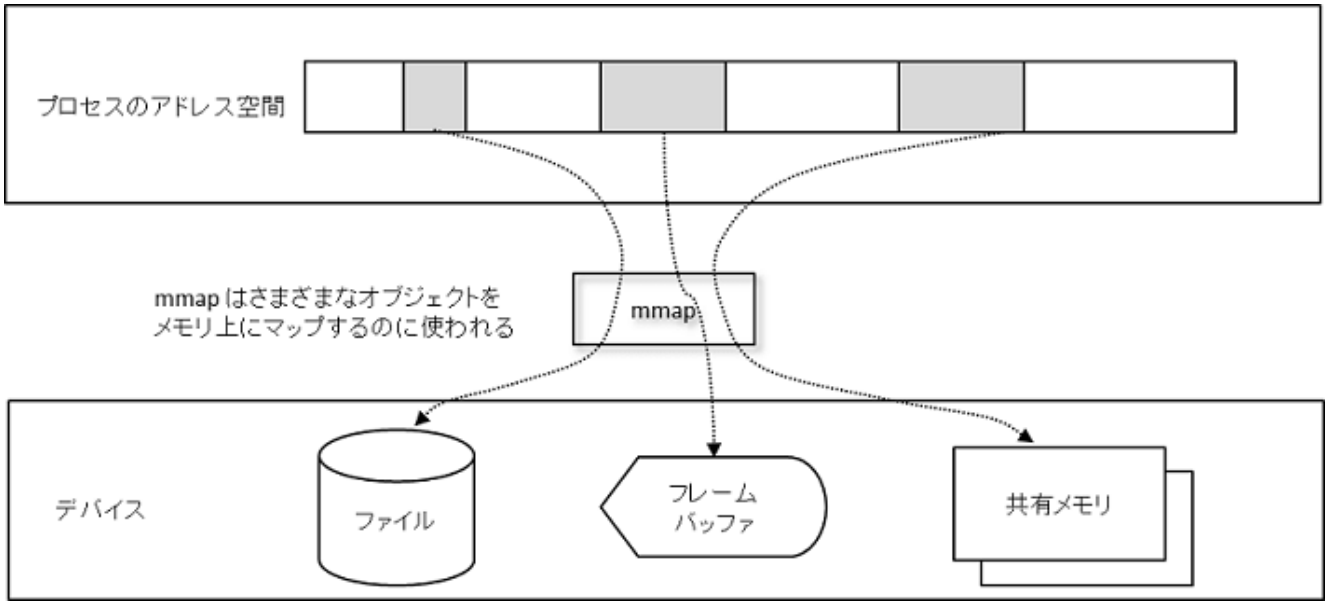

図 4.11.7　仮想ファイルとアドレス空間のマッピング

この枠組みでプロセス間でメモリを共有するとは、そのメモリ領域に対応する /dev/asmem のファイルディスクリプタを共有することに等しい[*39] 。そのため共有にかかるコストは小さくて済み、一旦プロセスのアドレス空間上にマップされた後は通常のメモリアクセスと等価な効率でのアクセスが可能となる。

4.11.4.2 新規インストールアプリでの **sharedUserId** の無効化

Android には、Manifest ファイル中に定義する、sharedUserId と呼ばれる要素を用いて、複数のアプリ間で、宣言しているパーミッションの許可状態や、ファイルアクセスの許可を共通化することが可能である。これは、パラメータに同じ値を定義し、同一の開発者証明書でアプリに署名がなされていた場合、Android 端末中で、複数アプリに同じユーザ ID が割り振られるためである。

このパラメータは、API Level 29 にて非推奨となったが、移行方法が用意されていなかった。マニフェストファイル中から、該当パラメータを削除した場合、アプリの更新でエラーが出てしまう為、一度設定されると変更が出来ない状況であった。結果、非推奨となったにもかかわらず、新規インストールでも sharedUserId が設定されたままの状態であった。

Android 13 がインストールされた端末では、Manifest ファイル中に以下の要素を定義することで、新規インストールアプリで sharedUserId の影響を受けないようになった。ただし、あくまで新規インストールアプリであり、既存アプリの更新の際には、sharedUserId で設定されたユーザ ID を利用し続ける。

[*39] ファイルディスクリプタはプロセス内で固有の値であるため、他のプロセスにそれを渡す場合は適切な変換が必要となるが、Android SDK API レベルではこれを気にする必要はない。

```
android:sharedUserMaxSdkVersion
```

この要素は、名称の通り、sharedUserId を適用する最大の端末バージョン (API Level) を指定する。本パラメータが有効なのは、Android 13 からであるため、執筆時点 (2022 年 7 月) では、有効に利用する場合は 32 を指定することになる。

4.12 NFC · FeliCa を利用する

NFC（Near Field Communication）と FeliCa は、どちらも「かざして通信する」近距離無線技術である。世界標準である NFC の枠組みの中に、日本固有の FeliCa（Type-F）が含まれる関係性にある。NFC には、通信方式や規格によって大きく分けて以下のタイプが存在する。

- Type-F (FeliCa): 主に日本国内で普及。Suica や PASMO などの交通系 IC カード、iD や QUICPay などの電子決済で広く利用されている。
- Type-A: 世界的に最も普及している規格。国内では taspo、海外の公衆電話や公共交通機関のカードで利用されている。
- Type-B: 堅牢なセキュリティを誇り、マイナンバーカード、運転免許証、パスポートなどで利用されている。
- NFC タッチ決済: Type-A/B を用いた国際標準の非接触決済。クレジットカード（Visa/Mastercard 等）や、Apple Pay、Google ウォレットなどのタッチ決済に利用される。

Android 端末における NFC の設定は、通常以下のメニュー等からユーザー自身が ON/OFF を切り替えることができる。
設定 > 接続済みのデバイス > 接続の設定 > NFC

4.12.1 サンプルコード

NFC/FeliCa の通信を行うアプリを開発する際、不要な規格のスキャンによる処理遅延の防止、非搭載端末でのアプリクラッシュや誤インストールの防止、および不要なバックグラウンド処理の抑制（バッテリーリーク防止）を考慮した、安全で効率的な実装設計が求められる。

ここでは、FeliCa（Type-F）カードを検出し、その基本情報である「IDm（製造 ID）」および「PMm（製造パラメータ）」を取得して表示するシンプルなリーダーアプリを例に、安全な NFC 機能の実装アプローチを解説する。

ポイント：

1. リーダーモード（enableReaderMode）を開始する際は、対象外の規格を余分に検出しないよう、必要な規格（FeliCa のみを対象とする場合は FLAG_READER_NFC_F）だけを指定する。
2. NFC 機能の利用がアプリの動作に必須である場合は、Google Play ストアの配信フィルタリング機能を利用して非搭載端末を事前に除外し、誤インストールを防ぐ。
3. アプリが画面の前面にある状態（onResume）でのみリーダーモードを有効化し、背面へ移行する際（onPause）には確実に解除（disableReaderMode）する。

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">
```

(次のページに続く)

(前のページからの続き)

```
    <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
            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.
```

(次のページに続く)

(前のページからの続き)

```
 */

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
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()

        // 端末のデフォルト NfcAdapter を取得
        nfcAdapter = NfcAdapter.getDefaultAdapter(this)
        if (nfcAdapter == null) {
            // ★ポイント 2★ USB 経由など、NFC 非搭載端末に直接 apk インストールされた場合のエラーハンド
```

(次のページに続く)

(前のページからの続き)

```
リング
            Toast.makeText(this, "NFC がサポートされていません", Toast.LENGTH_LONG).show()
        }

        setContent {
            // コピペ再利用を容易にするため、汎用的な MaterialTheme を使用
            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()

            // ★ポイント 1★ 検出規格を FeliCa（FLAG_READER_NFC_F）に限定し、NDEF チェック等の不要な
処理もスキップ設定する
            // ★ポイント 3★ アプリが前面にある状態（onResume）でリーダーモードを有効化
            it.enableReaderMode(
                this,
                this,
                NfcAdapter.FLAG_READER_NFC_F or NfcAdapter.FLAG_READER_SKIP_NDEF_CHECK,
                options
            )
        }
    }

    override fun onPause() {
        super.onPause()
        // ★ポイント 3★ アプリが背面に移動する際（onPause）には確実にリーダーモードを解除
        nfcAdapter?.disableReaderMode(this)
        isScanning = false
    }

    // カード検出時に別スレッドから呼び出されるコールバック
    override fun onTagDiscovered(tag: Tag) {
```

(次のページに続く)

(前のページからの続き)

```
        val nfcF = NfcF.get(tag)
        if (nfcF != null) {
            // FeliCa の固有データ（IDm、PMm）を安全に取得
            val idm = tag.id.toHexString()
            val pmm = nfcF.manufacturer.toHexString()
            cardInfo = CardInfo(idm = idm, pmm = pmm)
        }
    }

    // バイト配列を 16 進数文字列に変換する拡張関数
    private fun ByteArray.toHexString(): String {
        return joinToString("") { "%02X".format(it) }
    }
}

// 検出したカード情報を保持するデータクラス
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(
```

(次のページに続く)

(前のページからの続き)

```
                    text = if (isScanning) "カードをかざしてください" else "NFC が無効です",
                    style = MaterialTheme.typography.bodyLarge
                )
            }
        }
    }
}

@Composable
fun CardDetails(cardInfo: CardInfo) {
    Column(
        horizontalAlignment = Alignment.CenterHorizontally
    ) {
        InfoRow(label = "IDm (製造 ID)", value = cardInfo.idm)
        Spacer(modifier = Modifier.height(16.dp))
        InfoRow(label = "PMm (製造パラメータ)", 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 ルールブック

NFC / FeliCa 機能を利用する際には、以下のルールを遵守すること。

1. *NFC* 機能が動作に必須のアプリでは、*uses-feature* 属性で *required="true"* 宣言をして配信を制限する。(必須)
2. リーダーモード(*enableReaderMode*)は、*Activity* のライフサイクル(*onResume/onPause*)と完全に同期して有効化・無効化する(必須)
3. *enableReaderMode* を開始する際は、対象外の不要な通信規格(*Type-A/B/F*)を指定せず、ターゲットに適合したフラグのみを設定する(必須)
4. 取得したタグデータ(*IDm/PMm*、その他ブロックデータ等)を処理または外部サーバーに送信する際は、入力データの安全性を検証する(必須)

4.12.2.1 **NFC** 機能が動作に必須のアプリでは、**uses-feature** 属性で **required=”true”** 宣言をして配信を制限する。（必須）

アプリが NFC や FeliCa の機能を中核とし、それなしでは動作しないものである場合、NFC 非搭載端末へのインストールを防ぐために AndroidManifest.xml 上で必ず <uses-feature android:name=”android.hardware.nfc” android:required=”true” /> を宣言しなければならない。 これを怠ると、NFC 非搭載デバイスを使用しているユーザーがストアからアプリをインストールできてしまい、アプリ起動時や機能使用時に NullPointerException を起こしてクラッシュする原因となる。また、開発テスト中の直接インストール（USB デバッグ等）に備えて、コード上でも NfcAdapter.getDefaultAdapter(this) が null を返すケースを検知し、適切にハンドリング（警告メッセージを表示して処理を中断する等）を行うこと。

4.12.2.2 リーダーモード（**enableReaderMode**）は、**Activity** のライフサイクル（**onResume/onPause**）と完全に同期して有効化・無効化する（必須）

enableReaderMode による NFC デバイスの制御は、アプリがユーザーから直接見えるフォアグラウンドに位置する期間のみに限定しなければならない。具体的には、Activity のライフサイクルにおける onResume() でリーダーモードを有効化し、onPause() に入る際には確実に disableReaderMode() を呼び出して無効化する必要がある。 このライフサイクル同期を誤ると、アプリがバックグラウンドに回っている間も NFC 読み取りプロセスが常駐し続け、異常なバッテリー消費を引き起こしたり、他の NFC 機能を持つアプリの正常な動作（決済処理など）を阻害したりする原因となる。

4.12.2.3 **enableReaderMode** を開始する際は、対象外の不要な通信規格（**Type-A/B/F**）を指定せず、ターゲットに適合したフラグのみを設定する（必須）

enableReaderMode を呼び出す際には、接続対象とする通信規格に適合するフラグ（FeliCa であれば FLAG_READER_NFC_F）を明示的に指定し、不要な規格のスキャンをオプトアウトしなければならない。また、迅速な読み取りとリソース消費の抑制のため、NDEF（NFC Data Exchange Format）のメタデータ整合性チェックが不要なユースケースにおいては、 FLAG_READER_SKIP_NDEF_CHECK を併せて指定することが強く推奨される。 対象規格を制限せずにすべての規格をスキャン対象とした場合、センサーの過敏な反応や不必要なポーリングが発生し、意図しない規格のタグに反応してしまったり、スキャン処理自体のオーバーヘッドが大きくなりユーザー体験を損ねたりするリスクが高まる。

4.12.2.4 取得したタグデータ（**IDm/PMm**、その他ブロックデータ等）を処理または外部サーバーに送信する際は、入力データの安全性を検証する（必須）

NFC 接続によってカードから読み取られた IDm、PMm、あるいはセキュア/非セキュアエリアから読み出された各種データは、すべて外部から渡された「信頼できない入力データ（Input）」とみなさなければならない。 これらのデータをアプリ内部でパースして処理する場合や、Web サーバーとの連携処理に引き渡す場合、またはデータベース（SQLite）のクエリ条件として指定する場合は、SQL インジェクションやバッファオーバーフロー、制御文字の混入などを防ぐため、一般的なセキュアコーディングの原則に基づいた「入力データの安全性確認（Input Validation）」を必ず実施すること。

4.12.3 アドバンスト

4.12.3.1 **FeliCa** カードのセキュアデータ読み出しと改ざん対策の考慮事項

FeliCa カード（あるいはマイナンバーカードなどの Type-B カード）から単に固定された識別子である「IDm（製造 ID）」を取得してユーザー識別を行う手法は、非常に簡便なため多くの勤怠管理システムや受付システム等で採用されている。しかし、「IDm はカード表面に印字されたシリアル番号のような公開データであり、容易に成りすまし（偽造カードの作

成）が可能である」という根本的なセキュリティの癖を正しく理解しておく必要がある。

もしアプリで本人確認、キャッシュレス決済、重要エリアへのアクセス制限など、一定レベル以上のセキュリティが要求される処理を実装する場合は、公開情報である IDm のみに依存する設計は避け、以下のいずれかのアプローチを採用すること。

暗号キーを用いた相互認証と暗号化通信の実施

FeliCa の「セキュアエリア（システムコードやサービスコードで保護された領域）」にアクセスする際は、カードとリーダー（アプリ/サーバー）の間で暗号キーを用いた安全な相互認証プロセスを実行し、セッションごとに生成される一時鍵を用いてデータの読み書きを行う設計にする。

電子署名による真正性（完全性）の検証

マイナンバーカード（Type-B）から基本 4 情報を読み出す場合など、カードに記録されたデータ自体に付随する「公的個人認証サービス」の電子署名や、信頼できる発行元の署名（完全性）をアプリ/サーバー側で検証するロジックを実装し、成りすましや改ざんを確実に排除する。

5 セキュリティ機能の使い方

暗号や電子署名、Permission など、Android にはさまざまなセキュリティ機能が用意されている。これらのセキュリティ機能は取り扱いを間違えるとセキュリティ機能が十分に発揮されず抜け道ができてしまう。この章では開発者がセキュリティ機能を活用するシーンを想定した記事を扱う。

5.1 認証（パスワード・パスキー）画面を作る

従来の Android アプリ開発では、パスワード入力画面の UI を独自に実装し、入力された情報を自前で管理することが一般的であった。しかし現在では、セキュリティと利便性の両立を図るため、パスワードだけでなくパスキーや Google によるログインなどを単一の API で安全に扱う Credential Manager（認証情報マネージャー）の利用など、OS 標準の認証フレームワークへ委ねるモダンな認証アプローチが推奨されている。

本節では、レガシーなパスワード入力画面を実装する際に守るべき基本的な表示ルールに加え、機密情報の画面保護（スクリーンキャプチャの無効化）、および認証情報を扱う際のセキュアな設計方針について解説する。

5.1.1 サンプルコード

パスワードなどの認証情報を扱う画面を作る際、セキュリティ上考慮すべきポイントについて述べる。

ポイント：

1. パスワードを入力するときにはマスク表示（●で表示する）機能を用意する
2. パスワード等の機密画面では FLAG_SECURE を用いてスクリーンキャプチャを無効化する
3. アプリ内にパスワードや暗号シークレットをハードコーディングしない
4. Credential Manager 等の標準 API を利用し、安全に認証情報を扱う

以下のレイアウト XML では、パスワード入力欄にマスク表示を設定している。

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"
```

(次のページに続く)

(前のページからの続き)

```
    android:orientation="vertical"
    android:padding="32dp"
    android:gravity="center">

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="ログイン認証"
        android:textSize="24sp"
        android:textStyle="bold"
        android:layout_marginBottom="32dp" />

    <!-- ★ポイント 1★ android:inputType="textPassword" を設定することでマスク表示（●で表示する）
機能を用意している -->
    <EditText
        android:id="@+id/passwordEditText"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:hint="パスワード（または 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="認証する" />

</LinearLayout>
```

以下の Activity では、画面の保護設定と、モダンな認証 API を呼び出す際の基本的な考え方を示している。

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.
```

(次のページに続く)

(前のページからの続き)

```
 * 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?) {
        super.onCreate(savedInstanceState)

        // ★ポイント 2★ パスワード等の機密画面では FLAG_SECURE を用いてスクリーンキャプチャを無効化する
        // ※ Google Play ポリシーで強く要求されている必須のセキュリティ要件です。
        window.addFlags(WindowManager.LayoutParams.FLAG_SECURE)

        // レイアウトファイルは activity_password.xml を読み込みます
        setContentView(R.layout.activity_password)

        // ★ポイント 1★ XML 側で android:inputType="textPassword" を設定することでマスク表示（●で表示する）機能を用意している
        val passwordEditText = findViewById<EditText>(R.id.passwordEditText)
        val loginButton = findViewById<Button>(R.id.loginButton)

        loginButton.setOnClickListener {
            // ★ポイント 3★ アプリ内にパスワードや暗号シークレットをハードコーディングしない
            // 認証情報はユーザー入力や安全な KeyStore から取得し、ソースコード内に固定文字列として埋め込まないこと。

            // ★ポイント 4★ Credential Manager 等の標準 API を利用し、安全に認証情報を扱う
            // 独自の複雑な認証ロジックを自前で実装する代わりに、OS 標準の Credential Manager などを利用して
            // パスキーやパスワードの保存・取得フローを実行する。
            performAuthentication()
        }
    }

    private fun performAuthentication() {
        // ここに Credential Manager（GetCredentialRequest など）を呼び出す処理を実装します
```

(次のページに続く)

(前のページからの続き)

```
    }
}
```

5.1.2 ルールブック

認証画面を作る際、および認証情報を扱う際には以下のルールを守ること。

1. パスワードを入力するときにはマスク表示（●で表示する）機能を用意する （必須）
2. パスワード等の機密画面では *FLAG_SECURE* を用いてスクリーンキャプチャを無効化する（必須）
3. アプリ内にパスワードや暗号シークレットをハードコーディングしない（必須）
4. *Credential Manager* 等の標準 *API* を利用し、安全に認証情報を扱う（推奨）

5.1.2.1 パスワードを入力するときにはマスク表示（●で表示する）機能を用意する （必須）

スマートフォンは人混みで利用されることが多く、第三者にパスワードを盗み見られるリスクが大きい。アプリの仕様として、パスワードをマスク表示する機能が必要である。レイアウト XML で指定する場合は、 android:inputType 属性に ”textPassword” を指定する。動的にプログラム上で切り替える場合は、EditText クラスの setInputType() メソッドで InputType.TYPE_TEXT_VARIATION_PASSWORD を追加する。

5.1.2.2 パスワード等の機密画面では **FLAG_SECURE** を用いてスクリーンキャプチャを無効化する（必須）

パスワード入力画面ではパスワードなどの機密情報が画面上に表示される。そのような画面では、マルウェアによる画面録画や、第三者によるスクリーンキャプチャによって個人情報が流出してしまう恐れがあるため、スクリーンキャプチャを無効にしておく必要がある。

スクリーンキャプチャの無効化は、 WindowManager に addFlags(WindowManager.LayoutParams.FLAG_SECURE) を設定することで実現できる。Google Play のデベロッパープログラムポリシー（Device and Network Abuse）においても、セキュリティおよびプライバシー保護のため、他のアプリの FLAG_SECURE 設定を回避・迂回する手段の作成や促進を行うことは明確に禁止されている。機密画面での FLAG_SECURE の利用は、アプリとユーザーを守るための必須のセキュリティ要件である。

5.1.2.3 アプリ内にパスワードや暗号シークレットをハードコーディングしない（必須）

パスワード、API キー、暗号化鍵などのクレデンシャルをアプリのソースコードやアセットファイルにハードコーディングしてはならない。ハードコードされたシークレットは、リバースエンジニアリングツールを使うことで攻撃者に簡単に取得されてしまい、重大なセキュリティリスクを引き起こす。システムレベルの認証情報が必要な場合は KeyChain API を使用し、各アプリ専用の認証情報は Android KeyStore プロバイダを利用して保存することが必須である。

5.1.2.4 **Credential Manager** 等の標準 **API** を利用し、安全に認証情報を扱う（推奨）

パスワードを自前で保存したり、ダミー表示のロジックを複雑に作り込んだりするのではなく、パスキーや生体認証と連携できる Credential Manager 等の OS 標準の認証フレームワークを積極的に利用すべきである。標準 API に処理を委ねることで、セキュアな設計を保ったままユーザーの利便性を向上させることができる。

5.1.3 アドバンスト

5.1.3.1 ログイン処理について

パスワード入力が求められる場面の代表例はログイン処理である。ログイン処理で気を付けるポイントをいくつか紹介する。

ログイン失敗時のエラーメッセージ

ログイン処理では ID（アカウント）とパスワードの 2 つの情報を入力する。ログイン失敗時には ID が存在しない場合と、ID は存在するがパスワードが間違っている場合の 2 つがある。ログイン失敗のメッセージでこの 2 つの場合を区別して表示すると、攻撃者は「指定した ID が存在するか否か」を推測できてしまう。このような推測を許さないためにも、ログイン失敗時のメッセージでは、上記 2 つの場合を区別せずに下記のように表示すべきである。

メッセージ例： ログイン ID または パスワード が間違っています。

自動ログイン機能

一度、ログイン処理が成功すると次回以降はログイン ID とパスワードの入力を省略して、自動的にログインを行う機能がある。自動ログイン機能は煩わしい入力が省略できるので利便性が高まるが、その反面スマートフォンが盗難された場合に第三者に悪用されるリスクが伴う。

第三者に悪用された場合の被害が受け入れられる用途か、十分なセキュリティ対策が可能な場合にのみ、自動ログイン機能は利用することができる。例えば、オンラインバンキングアプリの場合、第三者に端末を操作されると金銭的な被害が出るので自動ログイン機能に合わせてセキュリティ対策が必須となる。対策としては、「決済処理などの金銭的な処理が発生する直前にはパスワードの再入力を求める」、「自動ログイン設定時にユーザーに対して十分に注意を喚起し、確実な端末のロックを促す」などいくつか考えられる。自動ログインの利用にあたっては、これらの対策を前提に利便性とリスクを勘案して、慎重な検討を行うべきである。

5.1.3.2 パスワード変更について

一度設定したパスワードを別のパスワードに変更する場合、以下の入力項目を画面上に用意すべきである。

- 現在のパスワード
- 新しいパスワード
- 新しいパスワード（入力確認用）

自動ログイン機能がついている場合、第三者がアプリを利用できる可能性がある。その場合、勝手にパスワードを変更されないよう、現在のパスワードの入力を求める必要がある。また、新しいパスワードが入力ミスで使用不能に陥る危険を減らすため、新しいパスワードは 2 回、入力を求める必要がある。

5.1.3.3 システムの「パスワードを表示」設定メニューについて

Android の設定メニューの中に「パスワードを表示」という設定がある。 設定 > セキュリティ > パスワードを表示

Android 5.0 より前のバージョンまで、この手順で設定できる。ただし Android 5.0 以降では、「パスワードを表示」はチェックボックスからトグルボタンに変更されている。

設定 ー パスワードを表示

「パスワードを表示」設定をオンにすると最後に入力した 1 文字が平文表示となる。一定時間（2 秒程度）経過後、または次の文字が入力されると平文表示されていた文字はマスク表示される。オフにすると、入力直後からマスク表示となる。これはシステム全体に影響する設定であり、EditText のパスワード表示機能を使用しているすべてのアプリに適用される。

パスワード入力時の文字の表示

5.1.3.4 画面録画の検出

Android 15 から、画面録画をアプリで検出する機能が追加された。

画面録画の実行中、アプリが可視または不可視に遷移される際にコールバックが呼び出される。登録されたプロセスの UID が所有するアクティビティが録画されている場合、アプリは可視であるとみなされる。アプリ上で機密性の高い操作を実行している際に、画面録画されていることをユーザに通知することが可能になった。

例えば、ユーザの操作や他のアプリによって画面の録画が行われている場合に、アプリ上で ID やパスワードを入力する画面や個人情報が表示される画面などが表示される前に、ユーザに対して警告を表示したりアプリの操作を禁止するなど意図しない機密情報・個人情報の録画や配信を防ぐことができる。

実装例は以下の通り。

AndroidManifest.xml

```
<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 認証情報マネージャーを自動入力と統合する

Android 15（および androidx.credentials ライブラリ）を利用することで、開発者は特定の View（例：ユーザー名やパスワードの入力フィールド）を認証情報マネージャー（Credential Manager）のリクエストに直接リンクさせることができる。ユーザーがこれらの View にフォーカスすると、対応するリクエストが認証情報マネージャーに送信され、生成された認証情報はプロバイダ間で集約され、自動入力 UI（キーボードのインライン候補やプルダウン候補など）に直接表示されるようになる。この機能は、ユーザーが認証情報マネージャーのアカウント選択ツールを誤って閉じた場合の代替手段としても利用可能である。

推奨エンドポイントとして Jetpack androidx.credentials ライブラリが以下の理由により推奨される：

- 統合性：androidx.credentials は Android Jetpack の一部であり、他の Jetpack ライブラリとシームレスに統合可能。
- 保守性：Google によって公式にサポートされ、定期的なアップデートやバグ修正が提供されるため、信頼性が高い。
- セキュリティ：認証情報の管理に関する最新のセキュリティベストプラクティスが実装されており、安全な認証

情報管理が可能。

- 簡便性：開発者が簡単に実装できるように設計されており、コードの複雑さを軽減。

実装手順

1. GetCredentialRequest の作成

```
val getPasswordOption = GetPasswordOption()
val getPublicKeyCredentialOption = GetPublicKeyCredentialOption(
    requestJson = requestJson
)
val getCredRequest = GetCredentialRequest(
    listOf(getPasswordOption, getPublicKeyCredentialOption)
)
```

2. getCredential API の呼び出し

```
coroutineScope.launch {
    try {
        val result = credentialManager.getCredential(
            context = activityContext, // Use an activity-based context
            request = getCredRequest
        )
        handleSignIn(result)
    } catch (GetCredentialException e) {
        handleFailure(e)
    }
}
```

3. ビューに自動入力機能を設定

```
import androidx.credentials.PendingGetCredentialRequest

usernameEditText.pendingGetCredentialRequest = PendingGetCredentialRequest(
    getCredRequest) { response -> handleSignIn(response)
}
passwordEditText.pendingGetCredentialRequest = PendingGetCredentialRequest(
    getCredRequest) { response -> handleSignIn(response)
}
```

注意事項

- setPendingGetCredentialRequest は androidx.credentials ライブラリの拡張 API であり、Kotlin と Java で呼び出し方法が異なる。
- 現在、この機能は View オブジェクトでのみ利用可能である。

5.1.3.6 物理デバイス使用時のパスワード非表示について

Android 17 では、セキュリティ強化の一環として、物理デバイス（USB や Bluetooth 接続の外付けキーボードなど）からパスワードを入力した際の表示挙動が変更されている。

従来の Android（Android 16 以前）では、パスワード入力欄に文字を入力すると、ユーザーの入力確認のためにシステムの「パスワードを表示」設定に従い「最後に入力した 1 文字だけが一瞬表示され、その後すぐに●（マスク）に変わる」仕組みが一般的であった。この挙動は画面上のソフトウェアキーボードだけでなく、物理キーボードでの入力時にも同様に適用されるケースがあり、ディスプレイの大きいタブレット端末等において周囲から覗き見られるリスク（ショルダーサーフィング）が高まっていた。

これに対して Android 17 では、リスクを低減するために挙動が見直され、物理キーボードからの入力時には、キー入力直後からすべての文字が即座に「●」などで表示され、パスワード文字を一時的にも表示しない（最初から完全にマスクする）ように変更された。

この表示挙動は Android 17 から導入された ShowSecretsSetting によって管理され、「タッチ入力（ソフトキーボード）」と「物理入力（ハードウェアキーボード）」でそれぞれ独立して表示制御される。開発者はアプリ側で独自のパスワード表示の仕組みを実装するのではなく、OS のセキュリティポリシーや設定 API（ shouldShowPhysicalInput() など）を尊重した標準の入力コンポーネントを利用することが重要である。

5.2 Permission と Protection Level

Permission の Protection Level には normal, dangerous, signature, signatureOrSystem の 4 種類がある。その他に「development」「system」「appop」も存在するが、一般的なアプリでは使用しないので本章での説明は省略する。Permission はどの Protection Level であるかによってそれぞれ、Normal Permission, Dangerous Permission, Signature Permission, SignatureOrSystem Permission と呼ばれる。以下、このような名称を使う。

5.2.1 サンプルコード

5.2.1.1 Android OS 既定の Permission を利用する方法

Android OS は電話帳や GPS などのユーザー資産をマルウェアから保護するための Permission というセキュリティの仕組みがある。Android OS が保護対象としている、こうした情報や機能にアクセスするアプリは、明示的にそれらにアクセスするための権限（Permission）を利用宣言しなければならない。ユーザー確認が必要な Permission では、その Permission を利用宣言したアプリがインストールされるときに次のようなユーザー確認画面が表示される[*1]。

[*1] Android 6.0(API Level 23) 以降では、ユーザー確認と権限の付与はインストール時に行われず、アプリの実行中に権限の利用を要求する仕様に変更された。詳細は「5.2.1.4. *Android 6.0 以降で Dangerous Permission を利用する方法*」および「5.2.3.6. *Android 6.0 以降の Permission モデルの仕様変更について*」を参照すること。

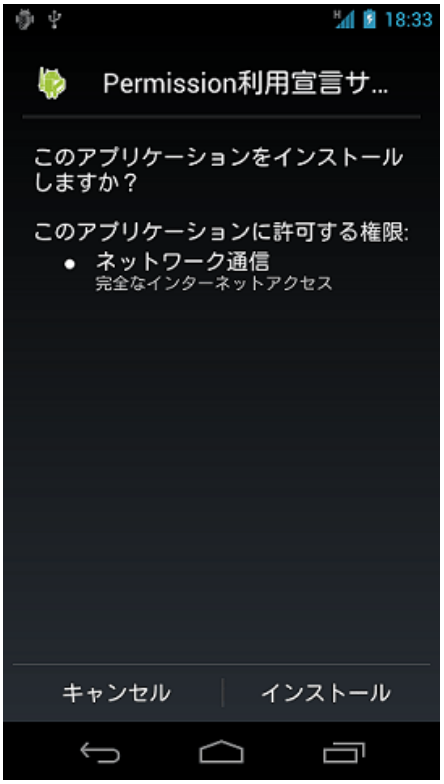

図 5.2.1　Permission 利用宣言確認画面

この確認画面により、ユーザーはそのアプリがどのような機能や情報にアクセスしようとしているのかを知ることができる。もし、アプリの動作に明らかに不必要な機能や情報にアクセスしようとしている場合は、そのアプリは悪意があるアプリの可能性が高い。ゆえに自分のアプリがマルウェアであると疑われないためにも、利用宣言する Permission は最小限にしなければならない。

ポイント：

1. 利用する Permission を AndroidManifest.xml に uses-permission で利用宣言する
2. 不必要な Permission は利用宣言しない

AndroidManifest.xml

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

  <!-- ★ポイント 1 ★　利用する Permission を AndroidManifest.xml に uses-permission で利用宣言する␣
↪-->
  <!-- インターネットにアクセスする Permission -->
  <uses-permission android:name="android.permission.INTERNET"/>

  <!-- ★ポイント 2 ★　不必要な Permission は利用宣言しない -->
  <!-- アプリの動作に不必要な Permission を利用宣言していると、ユーザーに不信感を与えてしまう -->

  <application
      android:allowBackup="false"
```

(次のページに続く)

（前のページからの続き）

```
      android:icon="@drawable/ic_launcher"
      android:label="@string/app_name" >
    <activity
        android:name=".MainActivity"
        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>
```

5.2.1.2 独自定義の Signature Permission で自社アプリ連携する方法

Android OS が定義する既定の Permission の他に、アプリが独自に Permission を定義することができる。独自定義の Permission（正確には独自定義の Signature Permission）を使えば、自社アプリだけが連携できる仕組みを作ることができる。複数の自社製アプリをインストールした場合に、それぞれのアプリの単機能に加え、アプリ間連携による複合機能を提供することで、複数の自社製アプリをシリーズ販売して収益を上げる、といった用途がある。

サンプルプログラム「独自定義 Signature Permission（UserApp）」はサンプルプログラム「独自定義 Signature Permission（ProtectedApp）」に startActivity() する。両アプリは同じ開発者鍵で署名されている必要がある。もし署名した開発者鍵が異なる場合は、UserApp は Intent を送信せず、ProtectedApp は受信した Intent を処理しない。またアドバンストセクションで説明しているインストール順序による Signature Permission 回避の問題にも対処している。

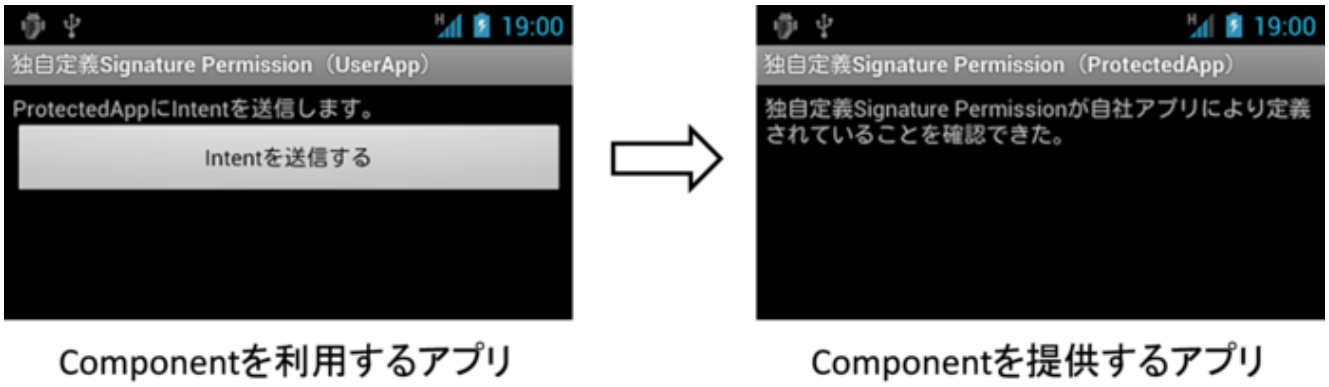

図 5.2.2　独自定義の Signature Permission で自社アプリを連携する

ポイント：Component を提供するアプリ

1. 独自 Permission を protectionLevel=”signature” で定義する
2. Component には permission 属性で独自 Permission 名を指定する
3. Component が Activity の場合には intent-filter を定義しない

4. ソースコード上で、独自定義 Signature Permission が自社アプリにより定義されていることを確認する
5. Android Studio からメニュー：Build -> Generated Signed APK と選択し、Component を利用するアプリと同じ開発者鍵で APK を署名する

AndroidManifest.xml

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

  <!-- ★ポイント 1 ★ 独自 Permission を protectionLevel="signature"で定義する -->
  <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" >

    <!-- ★ポイント 2 ★ Component には permission 属性で独自 Permission 名を指定する -->
    <activity
        android:name=".ProtectedActivity"
        android:exported="true"
        android:label="@string/app_name"
        android:permission="org.jssec.android.permission.protectedapp.MY_PERMISSION" >

      <!-- ★ポイント 3 ★ Component が Activity の場合には intent-filter を定義しない -->
    </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
```

(次のページに続く)

(前のページからの続き)

```
 * 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 {

    // 自社の Signature Permission
    private static final String MY_PERMISSION =
        "org.jssec.android.permission.protectedapp.MY_PERMISSION";

    // 自社の証明書のハッシュ値
    private static String sMyCertHash = null;
    private static String myCertHash(Context context) {
        if (sMyCertHash == null) {
            if (Utils.isDebuggable(context)) {
                // debug.keystore の"androiddebugkey"の証明書ハッシュ値
                sMyCertHash = "0EFB7236 328348A9 89718BAD DF57F544 D5CCB4AE B9DB34BC␣
↪1E29DD26 F77C8255";
            } else {
                // keystore の"my company key"の証明書ハッシュ値
                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);

        // ★ポイント 4★ ソースコード上で、独自定義 Signature Permission が自社アプリにより定義されて
```

(次のページに続く)

(前のページからの続き)

```
いることを確認する
        if (!SigPerm.test(this, MY_PERMISSION, myCertHash(this))) {
            mMessageView.setText("独自定義 Signature Permission が自社アプリにより定義されていな
い。");
            return;
        }

        // 証明書が一致する場合にのみ、処理を続行する
        mMessageView.setText("独自定義 Signature Permission が自社アプリにより定義されていることを確
認できた。");
    }
}
```

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;
```

(次のページに続く)

(前のページからの続き)

```
        correctHash = correctHash.replaceAll(" ", "");
        try{
            // sigPermName を定義したアプリのパッケージ名を取得する
            PackageManager pm = ctx.getPackageManager();
            PermissionInfo pi =
                pm.getPermissionInfo(sigPermName, PackageManager.GET_META_DATA);
            String pkgname = pi.packageName;
            // 非 Signature Permission の場合は失敗扱い
            if (pi.protectionLevel != PermissionInfo.PROTECTION_SIGNATURE) return false;
            // pkgname の実際のハッシュ値と正解のハッシュ値を比較する
            if (Build.VERSION.SDK_INT >= 28) {
                // ★ API Level >= 28 では Package Manager の API で直接検証が可能
                return pm.hasSigningCertificate(pkgname,
                                                Utils.hex2Bytes(correctHash),
                                                CERT_INPUT_SHA256);
            } else {
                // API Level < 28 の場合は PkgCert を利用し、ハッシュ値を取得して比較する
                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.
 * 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;
```

(次のページに続く)

(前のページからの続き)

```
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 は 32 バイト
        sha256 = sha256.toUpperCase();
        if (sha256.replaceAll("[0-9A-F]+", "").length() != 0)
            return false;    // 0-9A-F 以外の文字がある

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

    public boolean test(Context ctx, String pkgname) {
        // pkgname に対応する正解のハッシュ値を取得する
        String correctHash = mWhitelists.get(pkgname);

        // pkgname の実際のハッシュ値と正解のハッシュ値を比較する
        if (Build.VERSION.SDK_INT >= 28) {
            // ★ API Level >= 28 では Package Manager の API で直接検証が可能
            PackageManager pm = ctx.getPackageManager();
            return pm.hasSigningCertificate(pkgname,
                                            Utils.hex2Bytes(correctHash),
                                            CERT_INPUT_SHA256);
        } else {
            // API Level < 28 の場合は PkgCert の機能を利用する
            return PkgCert.test(ctx, pkgname, correctHash);
        }
    }
}
```

★ポイント 5 ★ Android Studio からメニュー：Build -> Generated Signed APK と選択し、Component を提供するアプリ

と同じ開発者鍵で署名する

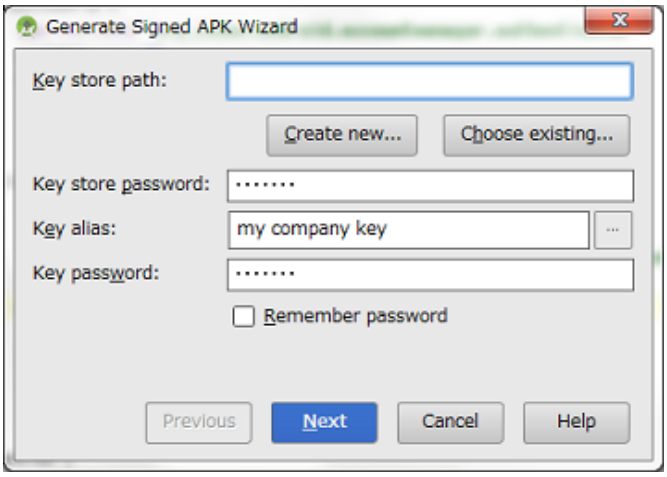

図 5.2.3　Component を提供するアプリと同じ開発者鍵で APK を署名する

ポイント：Component を利用するアプリ

6. 独自定義 Signature Permission は定義しない
7. uses-permission により独自 Permission を利用宣言する
8. ソースコード上で、独自定義 Signature Permission が自社アプリにより定義されていることを確認する
9. 利用先アプリが自社アプリであることを確認する
10. 利用先 Component が Activity の場合、明示的 Intent を使う
11. Android Studio からメニュー：Build -> Generated Signed APK と選択し、Component を提供するアプリと同じ開発者鍵で APK を署名する

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>

  <!-- ★ポイント 6★ 独自定義 Signature Permission は定義しない -->

  <!-- ★ポイント 7★ uses-permission により独自 Permission を利用宣言する -->
  <uses-permission
      android:name="org.jssec.android.permission.protectedapp.MY_PERMISSION" />

  <application
      android:allowBackup="false"
      android:icon="@drawable/ic_launcher"
```

(次のページに続く)

(前のページからの続き)

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

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 {
```

(次のページに続く)

(前のページからの続き)

```
    // 利用先の Activity 情報
    private static final String TARGET_PACKAGE  =
        "org.jssec.android.permission.protectedapp";
    private static final String TARGET_ACTIVITY =
        "org.jssec.android.permission.protectedapp.ProtectedActivity";

    // 自社の Signature Permission
    private static final String MY_PERMISSION =
        "org.jssec.android.permission.protectedapp.MY_PERMISSION";

    // 自社の証明書のハッシュ値
    private static String sMyCertHash = null;
    private static String myCertHash(Context context) {
        if (sMyCertHash == null) {
            if (Utils.isDebuggable(context)) {
                // debug.keystore の"androiddebugkey"の証明書ハッシュ値
                sMyCertHash = "0EFB7236 328348A9 89718BAD DF57F544 D5CCB4AE B9DB34BC␣
↪1E29DD26 F77C8255";
            } else {
                // keystore の"my company key"の証明書ハッシュ値
                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) {

        // ★ポイント 8★ ソースコード上で、独自定義 Signature Permission が自社アプリにより定義されて
いることを確認する
        if (!SigPerm.test(this, MY_PERMISSION, myCertHash(this))) {
            Toast.makeText(this,
                           "独自定義 Signature Permission が自社アプリにより定義されていない。",
                           Toast.LENGTH_LONG).show();
            return;
        }
```

(次のページに続く)

(前のページからの続き)

```
        // ★ポイント 9★ 利用先アプリが自社アプリであることを確認する
        if (!PkgCert.test(this, TARGET_PACKAGE, myCertHash(this))) {
            Toast.makeText(this,
                           "利用先アプリは自社アプリではない。",
                           Toast.LENGTH_LONG).show();
            return;
        }

        // ★ポイント 10★ 利用先 Component が Activity の場合、明示的 Intent を使う
        try {
            Intent intent = new Intent();
            intent.setClassName(TARGET_PACKAGE, TARGET_ACTIVITY);
            startActivity(intent);
        } catch(Exception e) {
            Toast.makeText(this,
                           String.format("例外発生:%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
 *
 *      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;
```

(次のページに続く)

(前のページからの続き)

```
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 は 32 バイト
        sha256 = sha256.toUpperCase();
        if (sha256.replaceAll("[0-9A-F]+", "").length() != 0)
            return false;    // 0-9A-F 以外の文字がある

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

    public boolean test(Context ctx, String pkgname) {
        // pkgname に対応する正解のハッシュ値を取得する
        String correctHash = mWhitelists.get(pkgname);

        // pkgname の実際のハッシュ値と正解のハッシュ値を比較する
        if (Build.VERSION.SDK_INT >= 28) {
            // ★ API Level >= 28 では Package Manager の API で直接検証が可能
            PackageManager pm = ctx.getPackageManager();
            return pm.hasSigningCertificate(pkgname,
                                            Utils.hex2Bytes(correctHash),
                                            CERT_INPUT_SHA256);
        } else {
            // API Level < 28 の場合は PkgCert の機能を利用する
            return PkgCert.test(ctx, pkgname, correctHash);
        }
    }
}
```

PkgCert.java

```
/*
 * Copyright (C) 2012-2026 Japan Smartphone Security Association
 *
```

(次のページに続く)

(前のページからの続き)

```
 * 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);
            if (pkginfo.signatures.length != 1) return null;  // 複数署名は扱わない
            Signature sig = pkginfo.signatures[0];
            byte[] cert = sig.toByteArray();
            byte[] sha256 = computeSha256(cert);
            return byte2hex(sha256);
        } catch (NameNotFoundException e) {
            return null;
```

(次のページに続く)

(前のページからの続き)

```
        }
    }

    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();
    }
}
```

★ポイント 11 ★ Android Studio からメニュー：Build -> Generated Signed APK と選択し、Component を提供するアプリと同じ開発者鍵で署名する

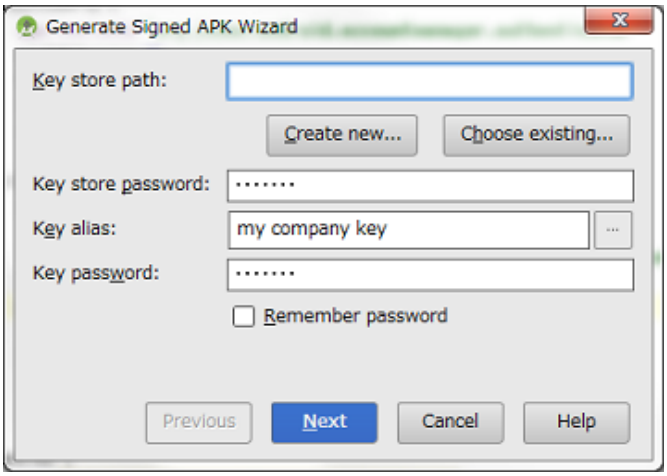

図 5.2.4　Component を提供するアプリと同じ開発者鍵で APK を署名する

Android 9.0 (API Level 28) 以降における署名の検証

Android 9.0 (API Level 28) で APK の署名スキーム V3 が導入され、署名用キーのローテーションができるようになった。これと同時にパッケージの署名関連の API もアップデートされている[*2]。アプリの署名検証の観点から変更点を見ると、PackageManager クラスの新規メソッド hasSigningCertificate() メソッドを使用して検証できるようになった。具体的には、ガイドのサンプルコード PkgCert クラスが行っていた、検証対象のパッケージから署名に使用した証明

[*2] 具体的な変更内容については Android Developers サイト https://developer.android.com/reference/android/content/pm/PackageManager を参照のこと。

書を取得しハッシュ値を計算するなどの処理を代替することができる。上に掲載したサンプルコードの SigPerm や PkgCertWhiteLists はこれを反映して API Level が 28 以上の場合に、この新規メソッド hasSigningCertificate() を用いている。hasSigningCertificate() は署名スキーム違いや複数署名に伴う検査の仕方の違いも吸収してくれるため、API Level 28 以降をターゲットとしている場合はこれを用いることをお薦めする[*3]。

5.2.1.3 アプリの証明書のハッシュ値を確認する方法

このガイド文書の各所で出てくるアプリの証明書のハッシュ値を確認する方法を紹介する。厳密には「APK を署名するときに使った開発者鍵の公開鍵証明書の SHA256 ハッシュ値」を確認する方法である。

Keytool により確認する方法

JDK に付属する keytool というプログラムを利用すると開発者鍵の公開鍵証明書のハッシュ値（証明書のフィンガープリントとも言う）を求めることができる。ハッシュ値にはハッシュアルゴリズムの違いにより MD5 や SHA1、SHA256 など様々なものがあるが、このガイド文書では暗号ビット長の安全性を考慮して SHA256 の利用を推奨している。残念なことに Android SDK で利用されている JDK6 に付属する keytool は SHA256 でのハッシュ値出力に対応しておらず、JDK7 以降に付属する keytool を使う必要がある。

Android のデバッグ証明書の内容を keytool で出力する例

```
> keytool -list -v -keystore <キーストアファイル> -storepass <パスワード>

キーストアのタイプ: jks
キーストア・プロバイダ: SUN

キーストアには 1 エントリが含まれます

別名: androiddebugkey
作成日: 2012/05/18
エントリ・タイプ: PrivateKeyEntry
証明書チェーンの長さ: 1
証明書 [1]:
所有者: CN=Android Debug, O=Android, C=US
発行者: CN=Android Debug, O=Android, C=US
シリアル番号: 4fb5d390
有効期間の開始日: Fri May 18 13:44:00 JST 2012 終了日: Tue Oct 04 13:44:00 JST 2039
証明書のフィンガプリント:
         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
署名アルゴリズム名: SHA1withRSA
サブジェクト公開鍵アルゴリズム: 1024 ビット RSA 鍵
バージョン: 3
```

(次のページに続く)

[*3] Android 9.0 (API Level 28) の android.content.pm.PackageManager に互換な Android Support Library は本記事を執筆時点でまだ提供されていない。

(前のページからの続き)

```
****************************************
****************************************
```

JSSEC 証明書ハッシュ値チェッカーにより確認する方法

JDK7 以降をインストールしなくても、JSSEC 証明書ハッシュ値チェッカーを使えば簡単に証明書ハッシュ値を確認できる。

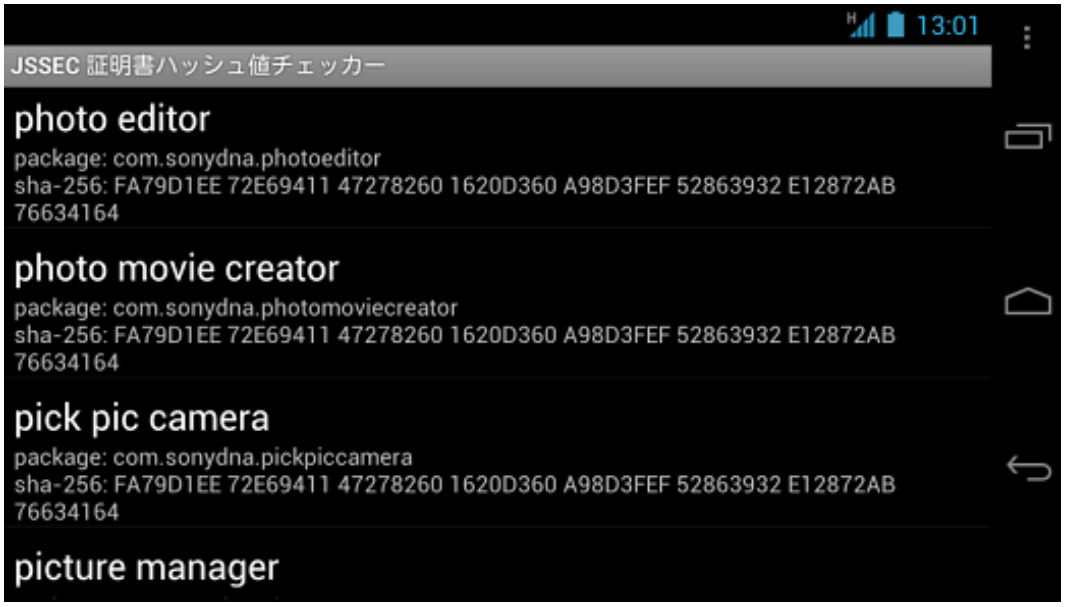

図 5.2.5　JSSEC 証明書ハッシュ値チェッカーによる確認

これは端末にインストールされているアプリの証明書ハッシュ値を一覧表示する Android アプリである。上図中、「sha-256」の右に表示されている 16 進数文字列 64 文字が証明書ハッシュ値である。このガイド文書と一緒に配布しているサンプルコードの「JSSEC CertHash Checker」フォルダがそのソースコード一式である。ビルドして活用していただきたい。

5.2.1.4 Android 6.0 以降で Dangerous Permission を利用する方法

アプリに対する Permission 付与のタイミングについて、Android 6.0(API Level 23) でアプリの実装にかかわる仕様変更が行われた。

Android 5.1(API Level 22) 以前の Permission モデル（「5.2.3.6. *Android 6.0 以降の Permission モデルの仕様変更について*」参照）では、アプリが利用宣言している Permission は全てアプリのインストール時に付与される。しかし、Android 6.0 以降では、Dangerous Permission についてはアプリが適切なタイミングで Permission を要求するよう、アプリ開発者が明示的に実装しなければならない。アプリが Permission を要求すると、Android OS はユーザーに対して下記のような確認画面を表示し、その Permission の利用を許可するかどうかの判断を求めることになる。ユーザーが Permission の利用を許可すれば、アプリはその Permission を必要とする処理を実行することができる。

図 5.2.6　Dangerous Permission 利用確認画面

Permission を付与する単位にも変更が加えられている。従来はすべての Permission が一括して付与されていたが、Android 6.0（API Level 23）以降では Permission Group 毎に、Android 8.0(API Level 26) 以降では Permission 個別に付与される。これに伴いユーザー確認画面も個別に表示され、ユーザーは Permission 利用の可・不可について従来よりも柔軟に判断できるようになった。アプリ開発者は、Permission の付与が拒否された場合も考慮して、アプリの仕様や設計を見直す必要がある。

Android 6.0 以降の Permission モデルについての詳細は「5.2.3.6. *Android 6.0 以降の Permission モデルの仕様変更について*」を参照すること。

ポイント：

1. アプリで利用する Permission を利用宣言する
2. 不必要な Permission は利用宣言しない
3. Permission がアプリに付与されているか確認する
4. Permission を要求する（ユーザーに許可を求めるダイアログを表示する）
5. Permission の利用が許可されていない場合の処理を実装する

AndroidManifest.xml

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

  <!-- ★ポイント 1 ★ アプリで利用する Permission を利用宣言する -->
  <!-- 連絡先情報を読み取る Permission (Protection Level: dangerous) -->
```

(次のページに続く)

(前のページからの続き)

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

  <!-- ★ポイント 2★　不必要な Permission は利用宣言しない -->

  <application
      android:allowBackup="true"
      android:icon="@mipmap/ic_launcher"
      android:label="@string/app_name"
      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.
 */
```

(次のページに続く)

(前のページからの続き)

```
package org.jssec.android.permission.permissionrequestingpermissionatruntime;

import android.Manifest;
import android.content.Intent;
import android.content.pm.PackageManager;
import android.os.Bundle;
import androidx.appcompat.app.AppCompatActivity;
import androidx.core.app.ActivityCompat;
import androidx.core.content.ContextCompat;
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);
        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() {
        // ★ポイント 3★ Permission がアプリに付与されているか確認する
        if (ContextCompat.checkSelfPermission(getApplicationContext(),
                                              Manifest.permission.READ_CONTACTS) !=
            PackageManager.PERMISSION_GRANTED) {
            // Permission が付与されていない
            // ★ポイント 4★ Permission を要求する (ユーザーに許可を求めるダイアログを表示する)
            ActivityCompat.requestPermissions(this,
                new String[]{Manifest.permission.READ_CONTACTS}, REQUEST_CODE_READ_
↪CONTACTS);
        } else {
            // Permission がすでに付与されている
            showContactList();
        }
    }
```

(次のページに続く)

（前のページからの続き）

```
    // ユーザー選択の結果を受けるコールバックメソッド
    @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) {
                // Permission の利用が許可されているので、連絡先情報を利用する処理を実行できる
                showContactList();
            } else {
                // Permission の利用が許可されていないため、連絡先情報を利用する処理は実行できない
                // ★ポイント 5★ Permission の利用が許可されていない場合の処理を実装する
                Toast.makeText(this, String.format("連絡先の利用が許可されていません"),
                               Toast.LENGTH_LONG).show();
            }
            return;
        }
    }

    // 連絡先一覧を表示
    private void showContactList() {
        // ContactListActivity を起動
        Intent intent = new Intent();
        intent.setClass(getApplicationContext(), ContactListActivity.class);
        startActivity(intent);
    }
}
```

5.2.2 ルールブック

独自 Permission 利用時には以下のルールを守ること。

1. *Android OS* 規定の *Dangerous Permission* はユーザーの資産を保護するためにだけ利用する （必須）
2. 独自定義の *Dangerous Permission* は利用してはならない （必須）
3. 独自定義 *Signature Permission* は *Component* の提供側アプリでのみ定義する （必須）
4. 独自定義 *Signature Permission* は自社アプリにより定義されていることを確認する （必須）
5. 独自定義の *Normal Permission* は利用してはならない （推奨）
6. 独自定義の *Permission* 名はアプリのパッケージ名を拡張した文字列にする （推奨）

5.2.2.1 Android OS 規定の Dangerous Permission はユーザーの資産を保護するためにだけ利用する （必須）

独自定義の Dangerous Permission の利用は非推奨（「5.2.2.2. 独自定義の *Dangerous Permission* は利用してはならない（必須）」参照）のため、ここでは Android OS 規定の Dangerous Permission を前提に話をする。

Dangerous Permission は他の 3 つの Permission と異なり、アプリにその権限を付与するかどうかをユーザーに判断を求める機能がある。Dangerous Permission を利用宣言しているアプリを端末にインストールするとき、次のような画面が表示される。これにより、そのアプリがどのような権限（Dangerous Permission および Normal Permission）を利用しようとしているのかをユーザーが知ることができる。ユーザーが「インストール」をタップすることで、そのアプリは利用宣言した権限が付与され、インストールされるようになっている。

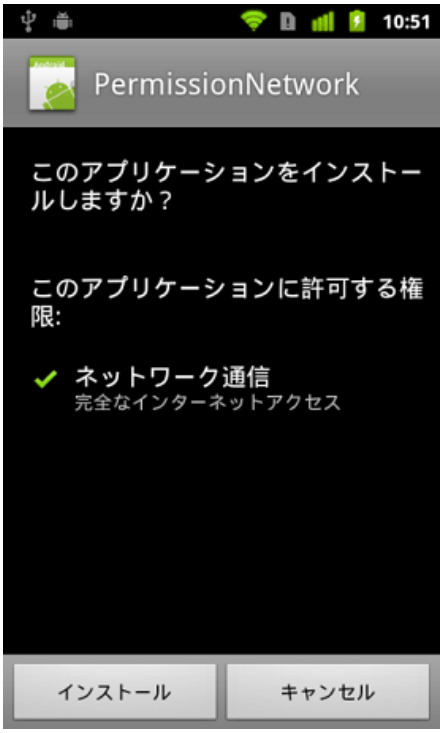

図 5.2.7　Android OS 規定の Dangerous Permission 利用確認画面

アプリの中には、ユーザーの資産とアプリ開発者が保護したい資産がある。それらのうち、Dangerous Permission で保護できるのはユーザーの資産だけであることに注意が必要である。なぜなら、権限の付与がユーザーの判断に委ねられているためである。一方、アプリ開発者が保護したい資産については、この方法では保護できない。

例えば、自社アプリだけと連携する Component は他社アプリからのアクセスを禁止したい場合を考える。このような Component を Dangerous Permission により保護するように実装したとする。他社アプリがインストールされるときに、ユーザーの判断により他社アプリに対して権限の付与を許可してしまうと、保護すべき自社の資産が他社アプリに悪用される危険が生じる。このような場合に自社の資産を保護するためには、独自定義の Signature Permission を使うとよい。

5.2.2.2 独自定義の Dangerous Permission は利用してはならない （必須）

独自定義の Dangerous Permission を使用しても、インストール時に「ユーザーに権限の許可を求める」画面が表示されない場合がある。つまり Dangerous Permission の特徴であるユーザーに判断を求める機能が働かないことがあるのだ。よって本ガイドでは「独自定義の Dangerous Permission を利用しない」ことをルールとする。

まず説明のために 2 つのアプリを想定する。1 つは独自の Dangerous Permission を定義し、この Permission により保護した Component を公開するアプリである。これを ProtectedApp とする。もう 1 つは ProtectedApp の Component を悪

用しようとする別のアプリでこれを AttackerApp とする。ここで AttackerApp は ProtectedApp が定義した Permission の利用宣言とともに同じ Permission の定義も行っているものとする。

AttackerApp がユーザーの許可なしに ProtectedApp の Component を利用できてしまうケースは以下のような場合に起きる。

1. ユーザーがまず AttackerApp をインストールすると、Dangerous Permission の利用許可を求める画面は表示されずに、そのままインストールが完了してしまう
2. 次に ProtectedApp をインストールすると、ここでも特に警告もなくインストールできてしまう
3. その後、ユーザーが AttackerApp を起動すると、AttackerApp はユーザーの気づかぬうちに ProtectedApp の Component にアクセスできてしまい、場合によっては被害に繋がる

このケースの原因は次の通りである。先に AttackerApp をインストールしようとすると uses-permission により利用宣言された Permission はまだその端末上では定義されていない。このとき Android OS はエラーとすることもなくインストールを続行してしまう。Dangerous Permission のユーザー確認はインストール時だけしか実施されないため、一度インストールされたアプリは権限を許可されたものとして扱われる。したがって後からインストールされるアプリの Component を同名の Dangerous Permission で保護していた場合、ユーザーの許可なく先にインストールされたアプリからその Component が利用できてしまうのである。

なお、Android OS 既定の Dangerous Permission はアプリがインストールされるときにはその存在が保証されているので、uses-permission しているアプリがインストールされるときには必ずユーザー確認画面が表示される。独自定義の Dangerous Permission の場合にだけこの問題は生じる。

現在、このケースで Component へのアクセスを防止するよい方法は見つかっていない。したがって、独自定義の Dangerous Permission は利用してはならない。

5.2.2.3 独自定義 **Signature Permission** は **Component** の提供側アプリでのみ定義する （必須）

自社アプリ間で連携する場合、実行時に Signature Permission をチェックすることでセキュリティを担保できることを「5.2.1.2. 独自定義の *Signature Permission* で自社アプリ連携する方法」で例示した。この仕組みを利用する際には、Protection Level が Signature の独自 Permission の定義は、Component 提供側アプリの AndroidManifest.xml でのみ行い、利用側アプリでは独自の Signature Permission を定義してはならない。

なお、signatureOrSystem Permission についても同様である。

以下がその理由となる。

提供側アプリより先にインストールされた利用側アプリが複数あり、どの利用側アプリも独自定義 Permission の利用宣言とともに Permission の定義もしている場合を考える。この状況で提供側アプリをインストールすると、すべての利用側アプリから提供側アプリにアクセスすることが可能になる。次に、最初にインストールした利用側アプリをアンインストールすると、Permission の定義が削除され、Permission が未定義となる。そのため、残った利用側アプリからの提供側アプリの利用が不可能となってしまう。

このように、利用側アプリで Permission の定義を行うと思わぬ Permission の未定義状態が発生するので、Permission の定義は保護する Component の提供側アプリのみ行い、利用側アプリで Permission を定義するのは避けなければならない。

こうすることで、提供側アプリのインストール時に権限付与が行われ、かつ、アンインストール時に Permission が未定義となり、提供側アプリと Permission の定義の存在期間が必ず一致するので、適正な Component の提供と保護が可能で

ある。なお、独自定義 Signature Permission に関しては、連携するアプリのインストール順によらず、利用側アプリに Permission 利用権限が付与されるため、この議論が成り立つことに注意[*4]。

5.2.2.4 独自定義 **Signature Permission** は自社アプリにより定義されていることを確認する （必須）

AnroidManifest.xml で Signature Permission を宣言し、Component をその Permission で保護しただけでは、実は保護が十分ではない。この詳細はアドバンストセクションの「5.2.3.1. 独自定義 *Signature Permission* を回避できる *Android OS* の特性とその対策」を参照すること。

以下、独自定義 Signature Permission を安全に正しく使う手順である。

まず、AndroidManifest.xml にて次を行う。

1. 保護したい Component のあるアプリの AndroidManifest.xml にて、独自 Signature Permission を定義する（Permission の定義）

   例： <permission android:name=”xxx” android:protectionLevel=”signature” />

2. 保護したい Component のある AndroidManifest.xml にて、その Component の定義タグの permission 属性で、独自定義 Signature Permission を指定する（Permission の要求宣言）

   例： <activity android:permission=”xxx” ... >...</activity>

3. 保護したい Component にアクセスする連携アプリの AndroidManifest.xml にて、uses-permission タグに独自定義 Signature Permission を指定する（Permission の利用宣言）

   例： <uses-permission android:name=”xxx” />

いて、ソースコード上にて次を実装する。

4. 保護したい Component でリクエストを処理する前に、独自定義した Signature Permission が自社アプリにより定義されたものかどうかを確認し、そうでなければリクエストを無視する（Component 提供側による保護）

5. 保護したい Component にアクセスする前に、独自定義した Signature Permission が自社アプリにより定義されたものかどうかを確認し、そうでなければ Component にアクセスしない（Component 利用側による保護）

最後に Android Studio の署名機能にて次を行う。

6. 連携するすべてのアプリの APK を同じ開発者鍵で署名する

ここで「独自定義した Signature Permission が、自社アプリにより定義されたものかどうかを確認」する必要があるが、具体的な実装方法についてはサンプルコードセクションの「5.2.1.2. 独自定義の *Signature Permission* で自社アプリ連携する方法」を参照すること。

なお、signatureOrSystem Permission についても同様である。

5.2.2.5 独自定義の **Normal Permission** は利用してはならない （推奨）

Normal Permission を利用するアプリは Android Manifest.xml に uses-permission で利用宣言するだけでその権限を得ることができる。そのため、一度インストールされてしまったマルウェアから Component を保護するような目的に Normal Permission は利用できない。

[*4] Normal/Dangerous Permission を利用する場合には、Permission が未定義のまま利用側アプリが先にインストールされると、利用側アプリへの権限の付与が行われず、提供側アプリがインストールされた後もアクセスができない

さらに独自定義 Normal Permission を用いてアプリ間連携を行う場合、連携する各アプリへの Permission の付与はインストール順に依存する。例えば、Permission を定義したコンポーネントを持つアプリよりも先にその Permission を利用宣言したアプリをインストールすると、Permission を定義したアプリをインストールした後も利用側アプリは Permission で保護されたコンポーネントにアクセスすることができない。

インストール順によりアプリ間連携ができなくなる問題を回避する方法として、連携する全てのアプリに Permission を定義することも考えられる。そうすることにより最初に利用側アプリがインストールされた場合でも、全ての利用側アプリが提供側アプリにアクセスすることが可能となる。しかし、最初にインストールした利用側アプリがアンインストールされた際に Permission が未定義な状態となり、他に利用側アプリが存在していても、それらのアプリから提供側アプリにアクセスすることができなくなってしまうのである。

以上のようにアプリの可用性が損なわれる恐れがあることから、独自定義 Normal Permission の利用は控えるべきである。

5.2.2.6 独自定義の **Permission** 名はアプリのパッケージ名を拡張した文字列にする （推奨）

複数のアプリが同じ名前で Permission を定義する場合、先にインストールされたアプリが定義する Protection Level が適用される。先にインストールされたアプリが Normal Permission を定義し、後にインストールされたアプリが同じ名前で Signature Permission を定義した場合、Signature Permission による保護がまったく効かない。悪意がない場合でも、複数のアプリにおいて Permission 名が衝突して意図しない Protection Level で動作する可能性がある。このような事故を防ぐため、Permission 名にはアプリのパッケージ名を入れた方が良い。

(パッケージ名).permission.(識別する文字列)

例えば、org.jssec.android.sample というパッケージに READ アクセスの Permission を定義するならば、次の様な命名が好ましい。

org.jssec.android.sample.permission.READ

5.2.3 アドバンスト

5.2.3.1 独自定義 **Signature Permission** を回避できる **Android OS** の特性とその対策

独自定義 Signature Permission は、同じ開発者鍵で署名されたアプリ間だけでアプリ間連携を実現する Permission である。開発者鍵はプライベート鍵であり絶対に公開してはならないものであるため、Signature Permission による保護は自社アプリだけで連携する場合に使われることが多い。

まずは、Android の Dev Guide https://developer.android.com/guide/topics/security/security.html で説明されている独自定義 Signature Permission の基本的な使い方を紹介する。ただし、後述するように、この使い方には Permission 回避の問題があることが分かっており、本ガイドに掲載した対策が必要となる。

以下、独自定義 Signature Permission の基本的な使い方である。

1. 保護したい Component のあるアプリの AndroidManifest.xml にて、独自 Signature Permission を定義する

   例： <permission android:name=”xxx” android:protectionLevel=”signature” />

2. 保護したい Component を持つアプリの AndroidManifest.xml で、保護したい Component に android:permission 属性を指定し、1. で定義した Signature Permission を要求する

   例： <activity android:permission=”xxx” ... >...</activity>

3. 保護したい Component にアクセスしたい連携アプリの AndroidManifest.xml にて、独自定義 Signature Permission を利用宣言する

   例： <uses-permission android:name=”xxx” />

4. 連携するすべてのアプリの APK を同じ開発者鍵で署名する

実は、この使い方だけでは、次の条件が成立すると Signature Permission 回避の抜け道ができてしまう。

説明のために独自定義の Signature Permission で保護したアプリを ProtectedApp とし、ProtectedApp とは異なる開発者鍵で署名したアプリを AttackerApp とする。ここで Signature Permission 回避の抜け道とは、AttackerApp は署名が一致していないにもかかわらず、ProtectedApp の Component にアクセス可能になることである。

条件 1. AttackerApp も ProtectedApp が独自定義した Signature Permission と同じ名前で Normal Permission を定義する（厳密には Signature Permission でも構わない）

例: <permission android:name=” xxx ” android:protectionLevel=”normal” />

条件 2. AttackerApp は独自定義した Normal Permission を uses-permission で利用宣言する

例: <uses-permission android:name=”xxx” />

条件 3. AttackerApp を ProtectedApp より先に端末にインストールする

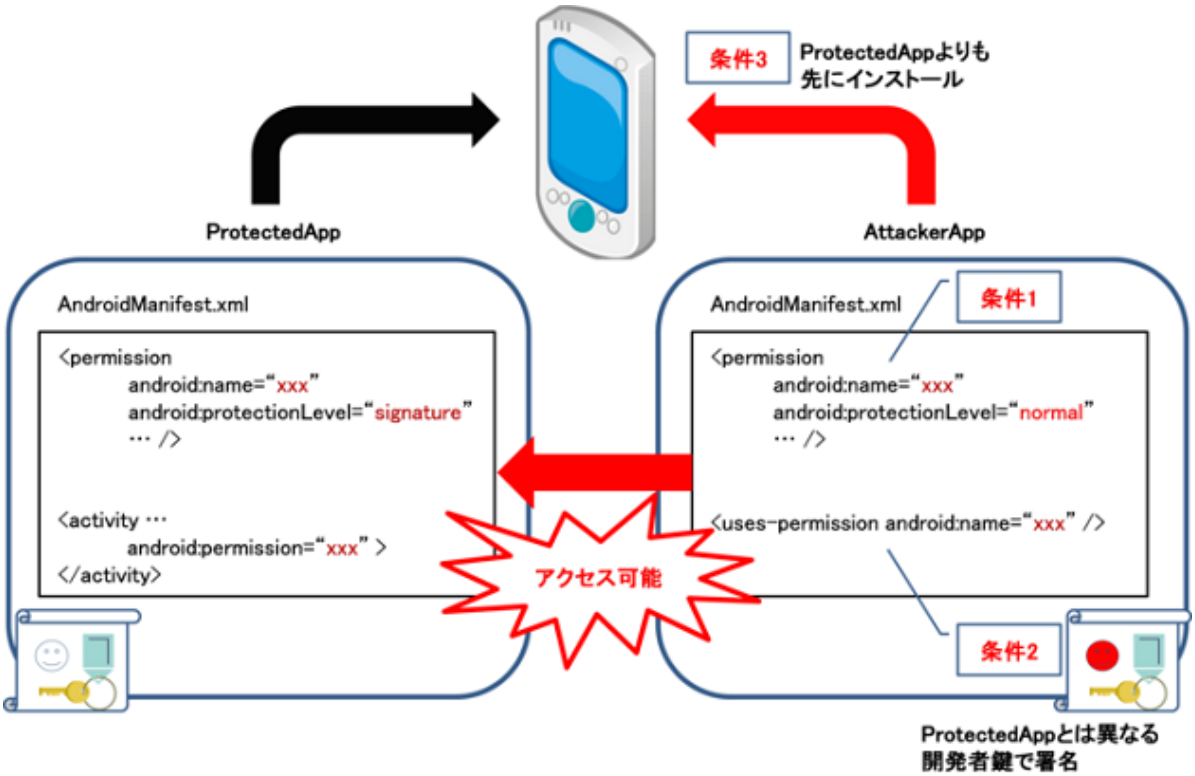

図 5.2.8　Signature Permission 回避の抜け道

条件 1 および条件 2 の成立に必要な ProtectedApp 独自定義の Permission 名は、APK ファイルから AndroidManifest.xml

を取り出せば攻撃者にとって容易に知ることができる。条件 3 もユーザーを騙すなどの方法により攻撃者にある程度制御の余地がある。

このように独自定義の Signature Permission には基本的な使い方だけでは保護を回避されてしまう危険性があり、抜け道をふさぐような対策が必要である。具体的にはルールセクションの「5.2.2.4. 独自定義 *Signature Permission* は自社アプリにより定義されていることを確認する （必須）」に掲載している方法で対処できるので、そちらを参照のこと。

5.2.3.2 ユーザーが AndroidManifest.xml を改ざんする

独自 Permission の Protection Level が意図しないものになるケースは既に説明した。そのことによる不具合を防ぐために、Java のソースコード側で何らかの対応を実施する必要があった。ここでは、AndroidManifest.xml が改ざんされるという視点から、ソースコード側の対応について述べる。改ざんを検知する簡易な実装例を提示するが、犯罪意識をもって改ざんを行うプロのハッカーに対してはほとんど効果がない方法であることに注意すること。

この節はアプリの改ざんに関するものであり、ユーザー自身が悪意を持っているケースである。本来はガイドラインの範囲外であるが、Permission に関する事、これを行うツールがアプリとして公開されている事、から「プロでないハッカーに対する簡易な対策」として述べておくことにした。

Android アプリは、root 権限無しに改ざんできることを頭に置いておく必要がある。なぜなら、AndroidManifest.xml を変更して APK ファイルを再生成、署名するツールが配布されているためである。このツールを使用する事で、誰でも任意のアプリから Permission を削除することが可能になっている。

事例としては INTERNET Permission を取り除いた AndroidManifest.xml から別署名の APK を生成し、アプリに組み込まれた広告モジュールが動作しないようにするケースが多いようである。個人情報がどこかに送信されているかもしれない等の不安が払拭されるということで、この種のツールの存在を評価しているユーザーも存在する。このような行為は、アプリに組み込まれた広告が機能しなくなるため、広告収入を期待している開発者に対して金銭的被害を与える行動であるとも言える。ユーザーのほとんどは罪の意識無くこれらの行為を行っていると思われる。

インターネット Permission を uses-permission で宣言しているアプリが、実行時に自身の AndroidManifest.xml に記載されている Permission を確認する実装例を次に示す。

```
public class CheckPermissionActivity extends Activity {

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

        // AndroidManifest.xml に定義した Permission を取得
        List<String> list = getDefinedPermissionList();

        // 改ざんを検知する
        if( checkPermissions(list) ){
            // OK
            Log.d("dbg", "OK.");
        }else{
            Log.d("dbg", "manifest file is stale.");
            finish();
```

（次のページに続く）

(前のページからの続き)

```
        }
    }

    /**
     * AndroidManifest.xml に定義した Permission をリストで取得する
     * @return
     */
    private List<String> getDefinedPermissionList(){
        List<String> list = new ArrayList<String>();
        list.add("android.permission.INTERNET");
        return list;
    }

    /**
     * Permission が変更されていないことを確認する。
     * @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)){
                        // 意図しない Permission が付加されている
                        return false;
                    }
                }
            }

            if(permissionList.size() == 0){
                // OK
                return true;
            }

        } catch (NameNotFoundException e) {
        }

        return false;
    }
}
```

5.2.3.3 APK の改ざんを検出する

「5.2.3.2. ユーザーが *AndroidManifest.xml* を改ざんする」ではユーザーによる Permission 改ざんの検出について説明した。しかし、アプリの改ざんは Permission に限らず、リソースを差し替えて別のアプリとしてマーケットで配布するなど、ソースコードを変更することなく改ざんし流用する事例が多様に存在する。ここでは APK ファイルが改ざんされたことを検出するためのより汎用的な方法を紹介する。

APK の改ざんを行うには、APK ファイルを一度展開し、内容を改変した後に再び APK ファイルとして再構成する必要がある。その際に改ざん者は元の開発者の鍵を持ち得ないので、改ざん者自身の鍵で APK を署名することになる。このように APK の改ざんには署名 (証明書) の変更を伴うため、アプリ起動時に APK の証明書と予めソースコードに埋め込んだ開発者の証明書を比較することで改ざんの有無を検出することができる。

以下にサンプルコードを示す。なお、実装例のままではプロのハッカーであれば改ざん検出の無効化が容易である。あくまで簡易な実装例であることを念頭においてアプリへの適用を検討するべきである。

ポイント：

1. 主要な処理を行うまでの間に、アプリの証明書が開発者の証明書であることを確認する

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 {
    // 自己証明書のハッシュ値
```

(次のページに続く)

(前のページからの続き)

```
    private static String sMyCertHash = null;
    private static String myCertHash(Context context) {
        if (sMyCertHash == null) {
            if (Utils.isDebuggable(context)) {
                // debug.keystore の"androiddebugkey"の証明書ハッシュ値
                sMyCertHash = "0EFB7236 328348A9 89718BAD DF57F544 D5CCB4AE B9DB34BC␣
↪1E29DD26 F77C8255";
            } else {
                // keystore の"my company key"の証明書ハッシュ値
                sMyCertHash = "D397D343 A5CBC10F 4EDDEB7C A10062DE 5690984F 1FB9E88B␣
↪D7B3A7C2 42E142CA";
            }
        }
        return sMyCertHash;
    }

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

        // ★ポイント 1 ★ 主要な処理を行うまでの間に、アプリの証明書が開発者の証明書であることを確認する
        if (!PkgCert.test(this, this.getPackageName(), myCertHash(this))) {
            Toast.makeText(this, "自己署名の照合　NG", Toast.LENGTH_LONG).show();
            finish();
            return;
        }
        Toast.makeText(this, "自己署名の照合　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.
```

(次のページに続く)

(前のページからの続き)

```
 * 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);
            if (pkginfo.signatures.length != 1) return null;  // 複数署名は扱わない
            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;
        }
```

(次のページに続く)

(前のページからの続き)

```
    }

    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 の再委譲問題

アプリが Android OS に保護されている電話帳や GPS といった情報や機能にアクセスするためには Permission を利用宣言しなければならない。Permission を利用宣言し許可されると、そのアプリにはその Permission が委譲されたことになり、その Permission により保護された情報や機能にアクセスできるようになる。

プログラムの組み方によっては、Permission を委譲された（許可された）アプリは Permission で保護されたデータを取得し、そのデータを別のアプリに何の Permission も要求せずに提供することもできてしまう。これは Permission を持たないアプリが Permission で保護されたデータにアクセスできることに他ならない。実質的に Permission を再委譲していることと等価になるので、これを Permission の再委譲問題と呼ぶ。このように Android の Permission セキュリティモデルでは、保護されたデータへのアプリからの直接アクセスだけしか権限管理ができないという仕様上の性質がある。

具体例を 図 5.2.9 に示す。中央のアプリは android.permission.READ_CONTACTS を利用宣言したアプリが連絡先情報を読み取って自分の DB に蓄積している。何の制限もなく Content Provider 経由で蓄積した情報を他のアプリに提供した場合に、Permission の再委譲問題が生じる。

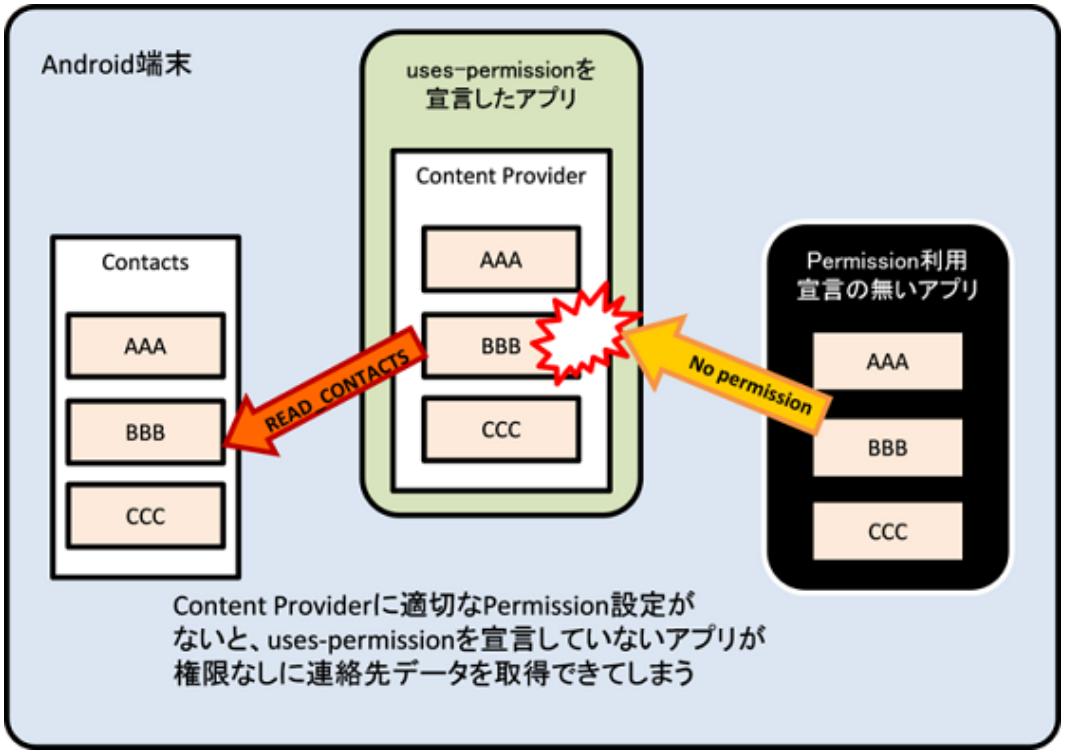

図 5.2.9　Permission を持たないアプリが連絡先情報を取得する

同様の例として、android.permission.CALL_PHONE を利用宣言したアプリが、同 Permission を利用宣言していない他のアプリからの任意の電話番号を受け付け、ユーザーの確認もなくその番号に電話を掛けることができるならば、Permission の再委譲問題がある。

Permission の利用宣言をして得た情報資産・機能資産をほぼそのままの形で他のアプリに二次提供する場合には、提供先アプリに対し同じ Permission を要求するなどして、元の保護水準を維持しなければならない。また情報資産・機能資産の一部分のみを他のアプリに二次提供する場合には、その情報資産・機能資産の一部分が悪用されたときの被害度合に応じた適切な保護が必要である。たとえば前述と同様に同じ Permission を要求したり、ユーザーへ利用許諾を確認したり、「4.1.1.1. 非公開 *Activity* を作る・利用する」「4.1.1.4. 自社限定 *Activity* を作る・利用する」などを利用して対象アプリの制限を設けるなどの保護施策がある。

このような再委譲問題は Permission に限ったことではない。Android アプリでは、アプリに必要な情報･機能を他のアプリやネットワーク・記憶媒体から調達することが一般に行われている。提供元が Android アプリであれば Permission、ネットワークであればログイン、記憶媒体であればアクセス制限といったように、それぞれ調達する際に必要な権限や制限が存在することも多い。こうして調達した情報や機能をその所有者であるユーザーから二次的に他のアプリに提供したり、ネットワークや記憶媒体に転送する際には、ユーザーの意図に反した利用がないように慎重に検討してアプリに対策を施すべきである。必要に応じて、Permission の例と同様に提供先に対して権限の要求や使用の制限を行わなければならない。ユーザーへの利用許諾もその一環である。

以下では、READ_CONTACTS Permission を利用して連絡先 DB から一覧を取得したアプリが、情報提供先のアプリに対して同じ READ_CONTACTS Permission を要求する例を示す。

ポイント：

1. Manifest で提供元と同じ Permission を要求する

AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
```

(次のページに続く)

(前のページからの続き)

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
          package="org.jssec.android.permission.transferpermission" >

  <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>

    <!-- ★ポイント 1★ Manifest で提供元と同じ Permission を要求する -->
    <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>
```

アプリが複数の Permission を要求する必要がある場合は、上記の方法では解決することができない。ソースコード上で Context#checkCallingPermission() や PackageManager#checkPermission() を使用して、呼び出し元のアプリが Manifest ですべての Permission の利用宣言を行っているかどうかを確認することになる。

Activity の場合

```
public void onCreate(Bundle savedInstanceState) {
// ～省略～
    if (checkCallingPermission("android.permission.READ_CONTACTS") ==
            PackageManager.PERMISSION_GRANTED
        && checkCallingPermission("android.permission.WRITE_CONTACTS") ==
            PackageManager.PERMISSION_GRANTED) {
        // 呼び出し元が正しく Permission を利用宣言していた時の処理
         return;
```

(次のページに続く)

(前のページからの続き)

```
    }
    finish();
}
```

5.2.3.5 独自定義 Permission の署名チェック機構について (Android 5.0 以降)

Android 5.0(API Level 21) 以降の端末では、独自の Permission を定義したアプリにおいて以下のような条件に合致するとインストールに失敗するように仕様変更された。

1. 既に同名の Permission を定義したアプリがインストールされている
2. そのインストール済みのアプリと署名が一致しない

この仕様変更により、保護対象の機能 (Component) の提供側アプリと利用側アプリの双方で Permission を定義した場合には、同じ Permission を定義した署名の異なる他社アプリが両アプリと同時にインストールされるのを防ぐことができる。しかしながら、「5.2.2.3. 独自定義 *Signature Permission* は *Component* の提供側アプリでのみ定義する （必須）」で言及した通り、アプリのアンインストール操作などによって、双方のアプリに Permission を定義するとその Permission が意図せず未定義状態になる場合があるため、この仕様を自社の定義した Signature Permission が他アプリに定義されていないことのチェックに活用することはできないことが分かっている。

結果として、自社限定アプリで独自定義 Signature Permission を利用する場合は、引き続き「5.2.2.3. 独自定義 *Signature Permission* は *Component* の提供側アプリでのみ定義する （必須）」、「5.2.2.4. 独自定義 *Signature Permission* は自社アプリにより定義されていることを確認する （必須）」のルールを順守する必要がある。

5.2.3.6 Android 6.0 以降の Permission モデルの仕様変更について

Android 6.0(API Level 23) においてアプリの仕様や設計にも影響を及ぼす Permission モデルの仕様変更が行われた。本節では Android 6.0 以降の Permission モデルの概要を解説する。また Android 8.0 以降での変更点および、Android 11 以降での「1 回だけのアクセス許可」についても記載する。

権限の付与・取り消しのタイミング

ユーザー確認が必要な権限（Dangerous Permission）をアプリが利用宣言している場合（「5.2.2.1. *Android OS* 規定の *Dangerous Permission* はユーザーの資産を保護するためにだけ利用する （必須）」参照）、Android 5.1（API Level 22）以前の仕様では、アプリのインストール時にその権限一覧が表示され、ユーザーがすべての権限を許可することでインストールが行われる。この時点で、アプリが利用宣言している (Dangerous Permission 以外の権限を含め) 全ての権限はアプリに付与され、一度付与された権限はアプリが端末からアンインストールされるまで有効である。しかし Android 6.0 以降の仕様では、権限の付与はアプリの実行時に行う仕様となり、アプリのインストール時には権限の付与もユーザーへの確認も行われない。アプリは、Dangerous Permission を必要とする処理を実行する際、事前にその権限がアプリに付与されているかどうかを確認し、権限が付与されていない場合には Android OS に確認画面を表示させ、ユーザーに権限利用の許可を求める必要がある[*5]。ユーザーが確認画面で許可することでその権限はアプリに付与される。ただし、ユーザーは一度アプリに許可した権限 (Dangerous Permission) を、設定メニューを通じて任意のタイミングで取り消すことができる（図 5.2.10）ため、権限がアプリに付与されておらず必要な情報や機能にアクセスすることができない状況においても、アプリが異常な動作を起こすことがないよう適切な処理を実装する必要がある。

[*5] Normal Permission および Signature Permission は Android OS により自動的に付与されるため、ユーザー確認を行う必要はない。

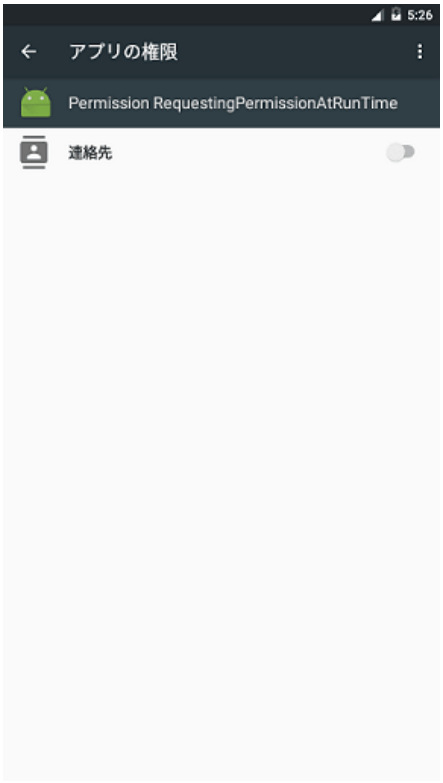

図 5.2.10　アプリの権限設定画面

また、Android11.0 より、位置情報、マイク、カメラに関連する一部のパーミッションについて、実行時に権限の付与を行う際に、「今回のみ」という選択肢を選べるようになった (1 回だけのアクセス許可)。この許可は、アプリの実行中のみ権限を有効化するためのもので、アプリが終了したり、バックグラウンドへ移動して一定時間後[*6] に権限が取り消される。対象となるパーミッションは以下の通りである。なお、Android6.0 以降の権限付与に対応できていれば、付与に際して特別な対応は不要である。

- android.permission.ACCESS_FINE_LOCATION
- android.permission.ACCESS_BACKGROUND_LOCATION
- android.permission.RECORD_AUDIO
- android.permission.CAMERA

権限の付与・取り消しの単位

いくつかの Permission はその機能や関連する情報の種類に応じて、Permission Group と呼ばれる単位でグループ化されている。例えば、カレンダー情報の読み取りに必要な Permission である android.permission.READ_CALENDAR と、カレンダー情報の書き込みに必要な Permission である android.permission.WRITE_CALENDAR は、どちらも android.permission-group.CALENDAR という Permission Group に属している。

Android 6.0（API Level 23）以降の Permission モデルにおいて、権限の付与や取り消しはこの Permission Group を単位として行われる。ただし、OS と SDK のバージョンの組み合わせによってこの単位が変わるので注意が必要となる（下記参照）。

- 端末：Android 6.0(API Level 23) 以降、アプリの targetSdkVersion：23~25 の場合

[*6] Android 11 で確認したところ、エミュレータ・実機 (pixel3) において、いずれも 1 分間であった。ただし、端末によって異なる可能性がある。

Manifest に android.permission.READ_CALENDAR と android.permission.WRITE_CALENDAR が記載されている状態で、アプリの実行時に android.permission.READ_CALENDAR の要求が行われ、ユーザーがこれを許可すると、Android OS は android.permission.READ_CALENDAR と android.permission.WRITE_CALENDAR の利用が両方とも許可されたとみなし権限が付与される。

- 端末：Android 8.0（API Level 26）以降、アプリの targetSdkVersion：26 以上の場合

要求した Permission の権限のみが付与される。つまり Manifest に android.permission.READ_CALENDAR と android.permission.WRITE_CALENDAR が記載されていても、android.permission.READ_CALENDAR のみを要求しユーザーに許可されたのなら android.permission.READ_CALENDAR の権限のみが付与される。ただし、その後 android.permission.WRITE_CALENDAR が要求された場合は、ユーザーに確認ダイアログが表示されることなく即時に権限が付与される[*7]。

また、権限の付与とは異なり、設定メニューからの権限の取り消しは Android 8.0 以降でも Permission Group 単位で行われる。Permission Group の分類については Developer Reference https://developer.android.com/intl/ja/guide/topics/security/permissions.html#perm-groups を参照すること。

仕様変更の影響範囲

アプリの実行時に Permission 要求が必要なのは、端末が Android 6.0 以降で動作していることに加え、アプリの targetSdkVersion が 23 以上に設定されている場合に限られる。端末が Android 5.1 以前で動作している場合や、アプリの targetSdkVersion が 23 未満である場合、権限は従来通りアプリのインストール時にまとめて付与される。ただし、アプリの targetSDKVersion が 23 未満であっても、端末が Android 6.0(API Level 23) 以降であれば、インストールされたアプリの Permission をユーザーが任意のタイミングで取り消すことができるため、意図しないアプリの異常終了が起きる可能性がある。早急に仕様変更に対応するか、アプリの maxSdkVersion を 22 以前に設定して、Android 6.0(API Level 23) 以降の端末にイントールされないようにするなどの対応が必要である。

また、Android 10 以降の端末では、Android 5.1(API Level 22) 以前の端末をターゲットにしているアプリを初めて実行したときに、正常に動作しない可能性がある旨の警告が表示される。更に、ユーザーにストレージアクセスなどの権限の付与を求めるアプリの場合、この警告の前に権限（許可する／許可しない）選択画面が表示される[*8]。

表 5.2.1　アプリへの権限付与のタイミング

端末の **Android OS** バージョン	アプリの **targetSDKVersion**	アプリへの権限付与のタイミング	ユーザーによる権限制御
≧ 8.0	≧ 26	アプリ実行時 (付与は Permission 単位)	あり
	＜ 26	アプリ実行時 (付与は Group 単位)	あり
	＜ 23	インストール時	あり (早急な対応が必要)
≧ 6.0	≧ 23	アプリ実行時 (付与は Group 単位)	あり
	＜ 23	インストール時	あり (早急な対応が必要)
≦ 5.1	≧ 23	インストール時	なし
	＜ 23	インストール時	なし

ただし、maxSdkVersion の効果は限定的であることに注意が必要である。maxSdkVersion を 22 以前に設定した場合、ア

[*7] この場合も、アプリによる android.permission.READ_CALENDAR と android.permission.WRITE_CALENDAR の利用宣言はともに必要である。

[*8] https://developer.android.com/about/versions/10/behavior-changes-all#low-target-sdk-warnings

プリを Google Play 経由で配布したときには、Android 6.0(API Level 23) 以降の端末が対象アプリのインストール可能端末としてリスト表示されなくなる。一方、Google Play 以外のマーケットプレイスでは maxSdkVersion の値がチェックされないことがあるため、Android 6.0(API Level 23) 以降の端末に対象アプリをインストールできる場合がある。

このように maxSdkVersion の効果は限定的であること、さらに Google が maxSdkVersion の使用を推奨していないことを踏まえ、早急に仕様変更に対応することをお勧めする。

なお、以下のネットワーク通信に関する Permission は、Android 6.0(API Level 23) 以降 Protection Level が dangerous から normal に変更されている。つまり、これらの Permission は利用宣言していても、ユーザーの明示的な許可を必要としないため、今回の仕様変更の影響を受けない。

- 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 Android 11 以降での使用していないアプリの権限が自動リセットされる機能について

Android 11(API Level 30) において、一定期間使用していないアプリの権限をリセットする機能が追加された。この期間は、targetSDKVersion によって、リセットの初期設定が異なる。本機能は、アプリの権限設定画面中の「アプリが使用されていない場合に権限を削除」によって設定する。

- targetSDKVersion=30:初期状態で有効
- targetSDKVersion<30:初期状態で無効

対象となるパーミッションは、Protection Level が Dangerous Permission のものである[*9]。ただし、1 回だけのアクセス許可の対象となるパーミッションでは、1 回だけを選択していた場合は毎回確認する設定のままとなる。

権限を必要とする機能を使う際に、都度パーミッションの確認していれば、エラー自体は発生しないが、例えばバックグラウンドで常に動き続けるアプリは、知らない間に機能が止まっていたという事態になる可能性がある。そのため、そのようなアプリでは、自動リセットを無効化するようにユーザに要求する必要がある。

1. PackageManager#isAutoRevokeWhitelisted() で自動リセット機能が無効になっているかを確認 (無効の場合は、true が返る)
2. 自動リセット機能が有効になっている場合は、アプリの設定画面を呼び出し、権限の設定画面へ誘導する。

ただし、isAutoRevokeWhitelisted() は API Level 30 で追加された API であるため、targetSDKVersion<30 のアプリでは判定できない。そのため、targetSDKVersion<30 のアプリで、自動リセットを無効化しておく必要があるアプリは、ユーザーにより自動リセットの設定が変更された場合に備えて、権限が取り消されていた場合に、再度権限の付与を要求するだけでなく、自動リセットの設定見直しを行う導線を設ける必要がある。

[*9] Android 11 で確認したところ、ACTIVITY_RECOGNITION(身体活動) については、自動リセットが行われても、権限は付与されたままであった。

5.2.3.8 実行時の権限の取り消し

Android 13 から、アプリはユーザによって付与された実行時の権限へのアクセスを、以下の API を利用することで取り消すことができるようになった。

- revokeSelfPermissionOnKill()
- revokeSelfPermissionsOnKill()

特定の実行時の権限を取り消すには、その権限の名前を revokeSelfPermissionOnKill() で指定する。複数の実行時の権限をまとめて指定するには、権限の名前をまとめて、revokeSelfPermissionsOnKill() に指定する。権限が付与されていない場合は、何も起きない。

これら API の導入は最小限の権限の原則に従ったものであり、アプリで定期的に実行時権限の見直しを行い、使わない権限については取り消しを行うことが望ましい。

サンプルコードは以下の通り。ここでは android.permission.CAMERA と android.permission.CALL_PHONE 権限を使用するアプリを例にする。

まずは宣言例。

```
<uses-permission android:name="android.permission.CAMERA" />
<uses-permission android:name="android.permission.CALL_PHONE" />
```

続いて、ボタン押下時に 2 つの権限をリクエストもしくは取り消すコード例を以下に示す。

```
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 -> {
```

(次のページに続く)

(前のページからの続き)

```
        revokeSelfPermissionsOnKill(Arrays.asList(PERMISSIONS));
    });
```

上記の例では 2 つの権限を同時に取り消しているが、個別に取り消す場合は以下のように revokeSelfPermissionOnKill を使用する。

```
revokeSelfPermissionOnKill(Manifest.permission.CAMERA);
```

revokeSelfPermissionOnKill または revokeSelfPermissionsOnKill を使用するアプリは Compile SDK 32 以下の場合ビルド不可のため、サンプルでは「@RequiresApi(api = 33)」を記述している。

実際に権限が取り消されるタイミングは非同期であり、システムが安全だと判断したタイミングで取り消される。すぐに取り消す場合はユーザーにアプリの再起動を促し System.exit() を実行するのが確実である。

5.2.3.9 SMS OTP（ワンタイムパスワード）の保護強化

近年、SIM スワップ（他人の電話番号を乗っ取る攻撃）やフィッシング（ユーザーを騙して OTP やパスワードを入力させる攻撃）、さらには SMS 転送型のマルウェアや Accessibility 権限の悪用などによって、SMS で送信された OTP（ワンタイムパスワード）が不正に取得される被害が問題となっている。この問題を背景に、Android 17 ではシステム制御下において SMS OTP へのアクセスを制限し、マルウェア等による認証情報の窃取を防止するための保護強化が導入された。

具体的には、アプリが READ_SMS や RECEIVE_SMS 権限を持っていても、OTP を含む SMS メッセージの受信後 3 時間が経過するまで、アプリから該当メッセージにアクセスできないように遅延される仕様となった。この 3 時間の遅延の間、 SMS_RECEIVED_ACTION ブロードキャストは保留され、SMS プロバイダに対するデータベースクエリからも該当メッセージはフィルタリングされる。デフォルトの SMS アプリなどはこの遅延の対象外となるが、従来のように READ_SMS 権限を用いて SMS の受信箱を直接監視し、プログラム的に OTP を自動取得していたアプリは大きな影響を受ける。

本保護機能の適用範囲は以下の通りである。

- すべてのアプリ（対象 **API** レベル問わず）: WebOTP 形式のメッセージに対して保護が適用される。ドメイン検証によりアプリが対象受信者と判断されない限り、メッセージへのアクセスが 3 時間遅延される。
- **Android 17** 以上をターゲットとするアプリ: WebOTP や SMS Retriever 形式以外の「標準の SMS メッセージ」に対しても保護が拡張され、同様に 3 時間遅延の対象となる。

推奨される実装の移行

アプリによる SMS 全体の読み取りを禁止し、不正な OTP 取得を防ぐという OS の方針に従い、OTP の抽出のために SMS メッセージの読み取りが必要なアプリは、権限不要の「SMS Retriever API」や、ユーザー同意を伴う「SMS User Consent API」といった Google が想定する正規 API への移行が強く推奨される。

表 5.2.2　推奨 API の特徴と比較

項目	SMS Retriever API	SMS User Consent API
ユーザー操作	不要（完全自動）	必須（1 タップ）
権限要件	不要	不要
SMS フォーマット	メッセージ末尾に 11 文字のアプリハッシュが必要	制限なし（任意のフォーマットで取得可能）
安定性と制御	高（フォーマットを自社で制御できれば安定して受信可能）	高（外部サービスの SMS でも取得可能）

自社で SMS 送信サーバーを制御でき、メッセージフォーマットにアプリハッシュを含めることができる場合は、完全自動でユーザー体験を損なわない SMS Retriever API が適している。一方で、外部サービスから送信される SMS などフォーマットの制御が難しい場合は、ユーザーの同意（1 タップ）を得ることでメッセージを 1 件だけ取得できる SMS User Consent API を利用するべきである。

OS の挙動変更の検証例

Android 17 で導入されたこの 3 時間遅延のブロック挙動は、従来の READ_SMS 権限を用いた SMS 読み取り処理を試作することで実際に確認できる。以下の検証コード抜粋は、ブロードキャスト（ SMS_RECEIVED_ACTION ）の受信と、コンテンツプロバイダ（ Telephony.Sms.CONTENT_URI ）へのクエリの両方を用いて SMS の読み取りを行おうとする例である。

ターゲット SDK を 36（Android 16）に設定した場合は、SMS 受信と同時に SMS 内の OTP が読み取れるのに対し、ターゲット SDK を 37（Android 17）に設定した場合は、SMS 内の OTP が即座に読み取れなくなる（3 時間遅延される）ことが確認できる。

```
// 1. ブロードキャスト（SMS_RECEIVED_ACTION）を受信する検証コード（抜粋）
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
            // 本文から OTP を抽出する処理
            // ※ Android 17（ターゲット SDK 37）では、受信後 3 時間経過するまでここが呼ばれない
        }
    }
}
```

```
// 2. コンテンツプロバイダから読み取る検証コード（抜粋）
val cursor = context.contentResolver.query(
    Telephony.Sms.CONTENT_URI,
    arrayOf(Telephony.Sms.BODY),
    null, null,
    Telephony.Sms.DATE + " DESC"
)
cursor?.use {
    if (it.moveToFirst()) {
```

（次のページに続く）

(前のページからの続き)

```
        val body = it.getString(0)
        // 本文から OTP を抽出する処理
        // ※ Android 17（ターゲット SDK 37）では、受信後 3 時間経過するまで該当メッセージが取得できない
    }
}
```

- 検証コードに関する注意**:** このコードは OS の挙動変化を確認するための検証用コードである。実際の商用アプリにおいては、前述の通り SMS Retriever API 等の正規 API へ移行する必要がある。また、検証を行う際、機密情報（SMS 本文や OTP）をログ出力（ Log.d など）しないように 4.8.2.1. 運用ログ情報にセンシティブな情報を含めない（必須） のルールに十分留意すること。

5.3 Account Manager に独自アカウントを追加する

Account Manager はアプリがオンラインサービスへアクセスするために必要となるアカウント情報（アカウント名、パスワード）および認証トークンを一元管理する Android OS の仕組みである[*10]。ユーザーは事前にアカウント情報を Account Manager に登録しておき、アプリがオンラインサービスにアクセスしようとしたときにユーザーの許可を得て、Account Manager がアプリに認証トークンを自動提供する仕組みである。パスワードという極めてセンシティブな情報をアプリが扱わなくて済むことが Account Manager の利点である。

Account Manager を使用したアカウント管理機能は 図 5.3.1 のような構成となる。「利用アプリ」は認証トークンの提供を受けてオンラインサービスにアクセスするアプリであり、前述のアプリのことである。一方、「Authenticator アプリ」は Account Manager の機能拡張であり、Authenticator と呼ばれるオブジェクトを Account Manager に提供することにより、Account Manager がそのオンラインサービスのアカウント情報および認証トークンを一元管理できるようになる。利用アプリと Authenticator アプリは別のアプリである必要はなく、一つのアプリとして実装することもできる。

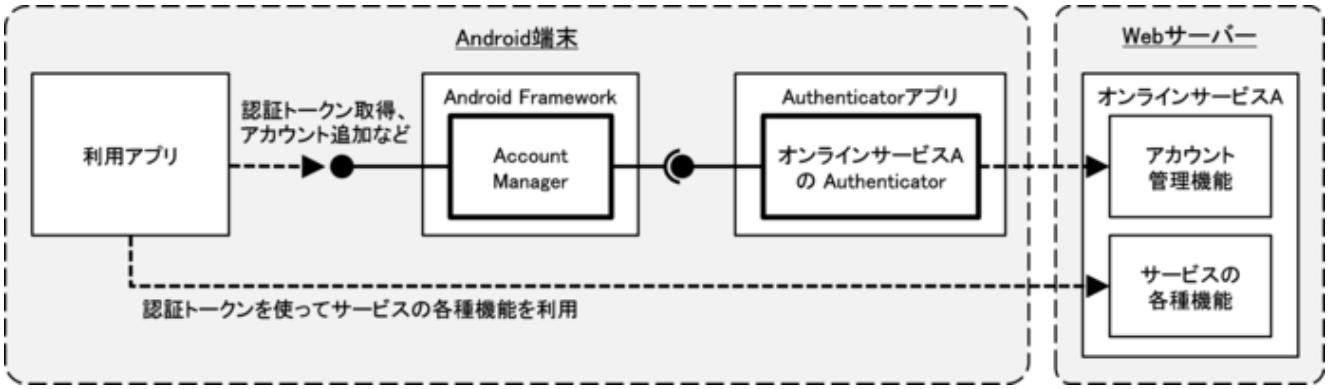

図 5.3.1　Manager を使用したアカウント管理機能の構成

5.3.1 サンプルコード

Authenticator アプリのサンプルとして「5.3.1.1. 独自アカウントを作る」を、利用アプリのサンプルとして「5.3.1.2. 独自アカウントを利用する」を用意した。JSSEC の Web サイトで配布しているサンプルコード一式ではそれぞれ AccountManager Authenticator および AccountManager User に対応している。

[*10] Account Manager はオンラインサービスとの同期の仕組みも提供するが、本節では扱っていない。

5.3.1.1 独自アカウントを作る

ここでは Account Manager が独自アカウントを扱えるようにする Authenticator アプリのサンプルコードを紹介する。このアプリはホーム画面から起動できる Activity は存在しない。もう一つのサンプルアプリ「5.3.1.2. 独自アカウントを利用する」から Account Manager 経由で間接的に呼び出されることに注意してほしい。

ポイント：

1. Authenticator を提供する Service は非公開 Service とする
2. ログイン画面 Activity は Authenticator アプリで実装する
3. ログイン画面 Activity は公開 Activity として他のアプリからの攻撃アクセスを想定する
4. KEY_INTENT には、ログイン画面 Activity のクラス名を指定した明示的 Intent を与える
5. アカウント情報や認証トークンなどのセンシティブな情報はログ出力しない
6. Account Manager にパスワードを保存しない
7. Authenticator とオンラインサービスとの通信は HTTPS で行う

AndroidManifest.xml にて Authenticator の IBinder を Account Manager に提供するサービスを定義。meta-data にて Authenticator を記述したリソース XML ファイルを指定。

AccountManager Authenticator/AndroidManifest.xml

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"

          xmlns:tools="http://schemas.android.com/tools">

  <!-- Authenticator を実装するのに必要な Permission -->
  <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" >

    <!-- Authenticator の IBinder を AccountManager に提供するサービス -->
    <!-- ★ポイント 1★ Authenticator を提供する Service は非公開 Service とする -->
    <service
        android:name=".AuthenticationService"
        android:exported="false" >
      <!-- intent-filter と meta-data はお決まりのパターン -->
      <intent-filter>
        <action android:name="android.accounts.AccountAuthenticator" />
      </intent-filter>
      <meta-data
          android:name="android.accounts.AccountAuthenticator"
          android:resource="@xml/authenticator" />
```

(次のページに続く)

(前のページからの続き)

```
    </service>

    <!-- アカウントを追加するときなどに表示されるログイン画面用の Activity -->
    <!-- ★ポイント 2★ ログイン画面 Activity は Authenticator アプリで実装する -->
    <!-- ★ポイント 3★ ログイン画面 Activity は公開 Activity として他のアプリからの攻撃アクセスを想定する -->
    <activity
        android:name=".LoginActivity"
        android:exported="true"
        android:label="@string/login_activity_title"
        android:theme="@android:style/Theme.Dialog"
        tools:ignore="ExportedActivity" />
  </application>

</manifest>
```

XML ファイルで Authenticator を定義。独自アカウントのアカウントタイプ等を指定する。

```
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" />
```

Authenticator のインスタンスを Account Manager に提供するサービス。このサンプルで実装する Authenticator である JssecAuthenticator クラスのインスタンスを onBind() で return するだけの簡単な実装でよい。

```
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
 * limitations under the License.
 */
```

(次のページに続く)

(前のページからの続き)

```
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();
    }
}
```

このサンプルで実装する Authenticator である JssecAuthenticator。AbstractAccountAuthenticator を継承して abstract メソッドをすべて実装する。これらのメソッドは Account Manager から呼ばれる。addAccount() および getAuthToken() では、オンラインサービスから認証トークンを取得するための LoginActivity を起動する intent を Account Manager に返している。

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;
```

(次のページに続く)

(前のページからの続き)

```
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;

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) {
            // 本サンプルコードではアカウントが既に存在する場合はエラーとする
            bundle.putString(AccountManager.KEY_ERROR_CODE, String.valueOf(-1));
            bundle.putString(AccountManager.KEY_ERROR_MESSAGE,
                             mContext.getString(R.string.error_account_exists));
        } else {
            // ★ポイント 2★ ログイン画面 Activity は Authenticator アプリで実装する
            // ★ポイント 4★ KEY_INTENT には、ログイン画面 Activity のクラス名を指定した明示的 Intent
を与える
            Intent intent = new Intent(mContext, LoginActivity.class);
            intent.putExtra(AccountManager.KEY_ACCOUNT_AUTHENTICATOR_RESPONSE, response);
```

(次のページに続く)

(前のページからの続き)

```
            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)) {
            // ★ポイント 4★ KEY_INTENT には、ログイン画面 Activity のクラス名を指定した明示的 Intent
を与える
            Intent intent = new Intent(mContext, LoginActivity.class);
            intent.putExtra(RE_AUTH_NAME, account.name);
            intent.putExtra(AccountManager.KEY_ACCOUNT_AUTHENTICATOR_RESPONSE, response);
            bundle.putParcelable(AccountManager.KEY_INTENT, intent);
        } else {
            // 指定されたアカウントが存在しない場合はエラーとする
            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;
    }
```

(次のページに続く)

(前のページからの続き)

```
    @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;
            }
        }
        return false;
    }
}
```

オンラインサービスにアカウント名、パスワードを送信してログイン認証を行い、その結果として認証トークンを取得する LoginActivity。新規アカウント追加および認証トークン再取得の場合に表示される。オンラインサービスへの実際のアクセスは WebService クラス内で実装されるものとしている。

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.
```

(次のページに続く)

(前のページからの続き)

```
 * 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);

        // アラートアイコン表示
        requestWindowFeature(Window.FEATURE_LEFT_ICON);
        setContentView(R.layout.login_activity);
        getWindow().setFeatureDrawableResource(Window.FEATURE_LEFT_ICON,
                                               android.R.drawable.ic_dialog_alert);

        // widget を見つけておく
        mNameEdit = (EditText) findViewById(R.id.username_edit);
        mPassEdit = (EditText) findViewById(R.id.password_edit);

        // ★ポイント 3★ ログイン画面 Activity は公開 Activity として他のアプリからの攻撃アクセスを想定
する
        // 外部入力は Intent#extras の String 型の RE_AUTH_NAME だけしか扱わない
        // この外部入力 String は TextEdit#setText()、WebService#login()、new Account() に
        // 引数として渡されるが、どんな文字列が与えられても問題が起きないことを確認している
```

(次のページに続く)

(前のページからの続き)

```
        mReAuthName = getIntent().getStringExtra(JssecAuthenticator.RE_AUTH_NAME);
        if (mReAuthName != null) {
            // ユーザー名指定で LoginActivity が呼び出されたので、ユーザー名を編集不可とする
            mNameEdit.setText(mReAuthName);
            mNameEdit.setInputType(InputType.TYPE_NULL);
            mNameEdit.setFocusable(false);
            mNameEdit.setEnabled(false);
        }
    }

    // ログインボタン押下時に実行される
    public void handleLogin(View view) {
        String name = mNameEdit.getText().toString();
        String pass = mPassEdit.getText().toString();

        if (TextUtils.isEmpty(name) || TextUtils.isEmpty(pass)) {
            // 入力値が不正である場合の処理
            setResult(RESULT_CANCELED);
            finish();
        }

        // 入力されたアカウント情報によりオンラインサービスにログインする
        WebService web = new WebService();
        String authToken = web.login(name, pass);
        if (TextUtils.isEmpty(authToken)) {
            // 認証が失敗した場合の処理
            setResult(RESULT_CANCELED);
            finish();
        }

        //  以下、ログイン成功時の処理

        // ★ポイント 5★ アカウント情報や認証トークンなどのセンシティブな情報はログ出力しない
        Log.i(TAG, "WebService login succeeded");


        if (mReAuthName == null) {
            // ログイン成功したアカウントを AccountManager に登録する
            // ★ポイント 6★ Account Manager にパスワードを保存しない
            AccountManager am = AccountManager.get(this);
            Account account = 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);
```

(次のページに続く)

(前のページからの続き)

```
            intent.putExtra(AccountManager.KEY_ACCOUNT_TYPE,
                            JssecAuthenticator.JSSEC_ACCOUNT_TYPE);
            setAccountAuthenticatorResult(intent.getExtras());
            setResult(RESULT_OK, intent);
        } else {
            // 認証トークンを返却する
            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();
    }
}
```

実際には WebService クラスはダミー実装となっており、常に認証が成功し固定文字列を認証トークンとして返すサンプル実装になっている。

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 {

    /**
     * オンラインサービスのアカウント管理機能にアクセスする想定
     *
```

(次のページに続く)

(前のページからの続き)

```
     * @param username アカウント名文字列
     * @param password パスワード文字列
     * @return 認証トークンを返す
     */
    public String login(String username, String password) {
        // ★ポイント 7★ Authenticator とオンラインサービスとの通信は HTTPS で行う
        // 実際には、サーバーとの通信処理を実装するが、 サンプルにつき割愛
        return getAuthToken(username, password);
    }

    private String getAuthToken(String username, String password) {
        // 実際にはサーバーから、ユニーク性と推測不可能性を保証された値を取得するが
        // サンプルにつき、通信は行わずに固定値を返す
        return "c2f981bda5f34f90c0419e171f60f45c";
    }
}
```

5.3.1.2 独自アカウントを利用する

独自アカウントの追加と認証トークンの取得を行うアプリのサンプルコードを以下に示す。もう一つのサンプルアプリ「5.3.1.1. 独自アカウントを作る」が端末にインストールされているときに、独自アカウントの追加や認証トークンの取得ができる。「アクセスリクエスト」画面は両アプリの署名鍵が異なる場合にだけ表示される。

図 5.3.2　サンプルアプリ AccountManager User の動作画面

ポイント：

1. Authenticator が正規のものであることを確認してからアカウント処理を実施する

利用アプリの AndroidManifest.xml。必要な Permission を利用宣言。必要な Permission については「5.3.3.1. *Account Manager の利用と Permission*」を参照。

```
AccountManager User/AndroidManifest.xml
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
```

(次のページに続く)

(前のページからの続き)

```
          >

  <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>

  <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>
```

利用アプリの Activity。画面上のボタンをタップすると addAcount() または getAuthToken() が実行される。指定のアカウントタイプに対応した Authenticator が偽物であるケースがあるので、正規の Authenticator であることを確認してからアカウント処理を始めていることに注意。

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.
```

(次のページに続く)

(前のページからの続き)

```
 * 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;
import android.os.Bundle;
import android.view.View;
import android.widget.TextView;

public class UserActivity extends Activity {

    // 利用する Authenticator の情報
    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("新しいアカウントを追加します");

        // ★ポイント 1★ Authenticator が正規のものであることを確認してからアカウント処理を実施する
        if (!checkAuthenticator()) return;
```

(次のページに続く)

(前のページからの続き)

```
    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("以下のアカウントを追加しました：");
                    logLine("    アカウント種別: %s", type);
                    logLine("    アカウント名: %s", name);
                } else {
                    String code = result.getString(AccountManager.KEY_ERROR_CODE);
                    String msg = result.getString(AccountManager.KEY_ERROR_MESSAGE);
                    logLine("アカウントが追加できませんでした");
                    logLine("  エラーコード %s: %s", code, msg);
                }
            } catch (OperationCanceledException e) {
            } catch (AuthenticatorException e) {
            } catch (IOException e) {
            }
        }
    },
null);
}

public void getAuthToken(View view) {
    logLine();
    logLine("トークンを取得します");

    // ★ポイント 1★ Authenticator が正規のものであることを確認してからアカウント処理を実施する
    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 {
```

(次のページに続く)

(前のページからの続き)

```
                    Bundle result = future.getResult();
                    String name = result.getString(AccountManager.KEY_ACCOUNT_NAME);
                    String authtoken = result.getString(AccountManager.KEY_AUTHTOKEN);
                    logLine("  %s さんのトークン:", name);
                    if (authtoken != null) {
                        logLine("    %s", authtoken);
                    } else {
                        logLine("    取得できませんでした");
                    }
                } catch (OperationCanceledException e) {
                    logLine("  例外: %s",e.getClass().getName());
                } catch (AuthenticatorException e) {
                    logLine("  例外: %s",e.getClass().getName());
                } catch (IOException e) {
                    logLine("  例外: %s",e.getClass().getName());
                }
            }
        },
        null);
    } else {
        logLine("アカウントが登録されていません");
    }
}

// ★ポイント 1 ★ Authenticator が正規のものであることを確認してからアカウント処理を実施する
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;
        }
    }

    if (pkgname == null) {
        logLine("Authenticator が見つかりません");
        return false;
    }

    logLine("  アカウントタイプ： %s", JSSEC_ACCOUNT_TYPE);
    logLine("  Authenticator のパッケージ名：");
    logLine("    %s", pkgname);

    if (!PkgCert.test(this, pkgname, getTrustedCertificateHash(this))) {
```

(次のページに続く)

(前のページからの続き)

```
            logLine("  正規の Authenticator ではありません（証明書不一致）");
            return false;
        }

        logLine("  正規の Authenticator です");
        return true;
    }

    // 正規の Authenticator アプリの証明書ハッシュ値
    // サンプルアプリ JSSEC CertHash Checker で証明書ハッシュ値は確認できる
    private String getTrustedCertificateHash(Context context) {
        if (Utils.isDebuggable(context)) {
            // debug.keystore の"androiddebugkey"の証明書ハッシュ値
            return "0EFB7236 328348A9 89718BAD DF57F544 D5CCB4AE B9DB34BC 1E29DD26 F77C8255
↪";
        } else {
            // keystore の"my company key"の証明書ハッシュ値
            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
 *
 * Licensed under the Apache License, Version 2.0 (the "License");
```

(次のページに続く)

(前のページからの続き)

```
 * 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);
            if (pkginfo.signatures.length != 1) return null;  // 複数署名は扱わない
            Signature sig = pkginfo.signatures[0];
            byte[] cert = sig.toByteArray();
            byte[] sha256 = computeSha256(cert);
            return byte2hex(sha256);
        } catch (NameNotFoundException e) {
            return null;
        }
```

(次のページに続く)

（前のページからの続き）

```
    }

    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();
    }
}
```

5.3.2 ルールブック

Authenticator アプリを実装する際には以下のルールを守ること。

1. *Authenticator* を提供する *Service* は非公開 *Service* とする （必須）
2. ログイン画面 *Activity* は *Authenticator* アプリで実装する （必須）
3. ログイン画面 *Activity* は公開 *Activity* として他のアプリからの攻撃アクセスを想定する （必須）
4. *KEY_INTENT* には、ログイン画面 *Activity* のクラス名を指定した明示的 *Intent* を与える （必須）
5. アカウント情報や認証トークンなどのセンシティブな情報はログ出力しない （必須）
6. *Account Manager* にパスワードを保存しない （推奨）
7. *Authenticator* とオンラインサービスとの通信は *HTTPS* で行う （必須）

利用アプリを実装する際には以下のルールを守ること。

8. *Authenticator* が正規のものであることを確認してからアカウント処理を実施する （必須）

5.3.2.1 Authenticator を提供する Service は非公開 Service とする （必須）

Authenticator を提供する Service は Account Manager から利用されることを前提としており、他のアプリがアクセスできてはならない。非公開 Service とすることにより、他のアプリからのアクセスを排除することができる。また Account Manager は system 権限で動作しているので非公開 Service であってもアクセスできる。

5.3.2.2 ログイン画面 **Activity** は **Authenticator** アプリで実装する （必須）

新規アカウント追加および認証トークン再取得の場合に表示されるログイン画面は Authenticator アプリで実装すべきである。利用アプリ側で独自にログイン画面を用意してはならない。この記事の冒頭で「パスワードという極めてセンシティブな情報をアプリが扱わなくて済むことが Account Manager の利点である。」と述べた。もし利用アプリ側でログイン画面を用意してしまうと、利用アプリがパスワードを扱ってしまうことになり、Account Manager の思想から逸脱した設計となってしまう。

Authenticator アプリがログイン画面を用意することにより、ログイン画面を操作できるのは端末のユーザーだけに限定される。これは悪意あるアプリが直接ログインを試みたり、アカウントを作成したりといったアカウント攻撃をする手段がないということである。

5.3.2.3 ログイン画面 **Activity** は公開 **Activity** として他のアプリからの攻撃アクセスを想定する （必須）

ログイン画面 Activity は利用アプリの権限で起動する仕組みとなっている。利用アプリと Authenticator アプリの署名鍵が異なる場合にもログイン画面 Activity が表示されるためには、ログイン画面 Activity は公開 Activity として実装しなければならない。

ログイン画面 Activity が公開 Activity であるということは、悪意あるアプリからも起動される可能性があるということである。入力データは一切信用してはならない。したがって「3.2. 入力データの安全性を確認する」で述べたような対策が必要となる。

5.3.2.4 **KEY_INTENT** には、ログイン画面 **Activity** のクラス名を指定した明示的 **Intent** を与える （必須）

Authenticator がログイン画面 Activity を開きたいときには、Account Manager に返す Bundle の中にログイン画面 Activity を起動する Intent を KEY_INTENT で与えることになっている。ここで与える Intent はログイン画面 Activity をクラス名で指定する明示的 Intent でなければならない。暗黙的 Intent を与えた場合は、フレームワークが Authenticator アプリがログイン画面のために用意した Activity 以外の Activity の起動を試みる可能性がある。これによって、Andorid 4.4（API Level 19）以降のバージョンではアプリがクラッシュしたり、Android 4.4（API Level 19）より前のバージョンでは他のアプリの用意した意図しない Activity が起動したりする場合がある。

Android 4.4（API Level 19）以降のバージョンでは、フレームワークが KEY_INTENT で与える Intent で起動されるアプリの署名と Authenticator アプリの署名が一致しない場合には SecurityException を発生させるため、偽のログイン画面を起動される恐れはないが、正規のログイン画面を起動できずユーザーの正常なアプリ利用を妨げられるおそれがある。Android 4.4（API Level 19）より前のバージョンでは、悪意のあるアプリの用意した偽のログイン画面を起動され、ユーザーが悪意のあるアプリにパスワード等認証情報を入力してしまう危険がある。よって、いずれにバージョンであっても、KEY_INTENT で与える Intent は明示的 Intent でなければならない。

5.3.2.5 アカウント情報や認証トークンなどのセンシティブな情報はログ出力しない （必須）

オンラインサービスに接続するアプリは、その開発時だけでなく運用時においても、オンラインサービスにうまく接続できないトラブルに悩まされることがある。接続できない原因は多岐に渡り、ネットワーク環境の整備不足、通信プロトコルの実装ミス、Permission 不足、認証エラーなど様々である。こうした原因の切り分けを目的として、プログラム内部で得られた情報をログ出力する実装もよくみられる。

パスワードや認証トークンなどのセンシティブな情報は決してログ出力してはならない。ログ情報は他のアプリからも読み取ることができるため情報漏洩の原因となりかねないからだ。アカウント名も漏洩も被害につながる場合にはログ出力してはならない。

5.3.2.6 Account Manager にパスワードを保存しない （推奨）

Account Manager に登録するアカウントには、パスワードと認証トークンの 2 つの認証情報を保存することができる。これらの情報は次のディレクトリの accounts.db の中に平文で（つまり暗号化されず）保存される。

- Android 4.1 以前

/data/system/accounts.db

- Android 4.2 以降 Android 6.0 以前

/data/system/users/0/accounts.db

- Android 7.0 以降

/data/system_ce/0/accounts_ce.db

※ Android 4.2 以降はマルチユーザー機能がサポートされているため、ユーザーに合わせたディレクトリへ保存されるように変更されている。また、Android 7.0 以降では Direct Boot 対応のため、ロック時にデータを扱う際のデータベース /data/system_de/0/accounts_de_db とアンロック時にデータを扱う /data/system_ce/0/accounts_ce.db にデータベースファイルが分割された。認証情報は平文の状態で後者のデータベースファイルに保存される。

この accounts.db の内容を読み取るためには root 権限または system 権限が必要であり、市販の Android 端末では読み取ることができない。もし、攻撃者に root 権限や system 権限が奪われてしまう脆弱性が Android OS にある場合には、accounts.db の中に保存された認証情報が危険にさらされることになる。

この記事で紹介している Authenticator アプリは、Account Manager に認証トークンは保存するが、ユーザーのパスワードは保存しない設計としている。一定の期間以内にオンラインサービスに継続的に接続していれば、認証トークンの有効期間が延長されるのが一般的であるため、パスワードを保存しない設計で十分であることが多い。

認証トークンは一般にパスワードよりも有効期限が短く、いつでも無効化できる特徴がある、いわば使い捨ての認証情報である。万一、認証トークンが漏洩したとしても、認証トークンを無効化することができるため、認証トークンはパスワードに比べ安全性が高いとされている。認証トークンが無効化された場合には、ユーザーはもう一度パスワードを入力して新しい認証トークンを取得すればよい。

パスワードが漏洩した場合、パスワードを無効化してしまうと、そのユーザーはオンラインサービスを利用できなくなってしまう。このような場合、コールセンター対応等が必要となってしまうため大きなコストが発生する。ゆえに Account Manager にパスワードを保存する設計はできるだけ避けるべきである。どうしてもパスワードを保存する設計をしなければならない場合は、パスワードを暗号化して、暗号化の鍵を難読化するなど、高度なリバースエンジニアリング対策を実施することになる。

5.3.2.7 Authenticator とオンラインサービスとの通信は HTTPS で行う （必須）

パスワードや認証トークンはいわゆる認証情報といい、これを第三者に奪われてしまうと、第三者がユーザーになりすましできることになる。Authenticator はオンラインサービスとこうした認証情報を送受信することになるので、HTTPS 等の安全性の確立した暗号化通信方式で通信しなければならない。

5.3.2.8 **Authenticator** が正規のものであることを確認してからアカウント処理を実施する （必須）

端末に同一のアカウントタイプを定義した Authenticator が複数存在する場合、先にインストールされた Authenticator が有効になる。自分の Authenticator が後にインストールされた場合には利用されないということである。

もし先にインストールされた Authenticator がマルウェアによる偽装であった場合には、ユーザーが入力したアカウント情報がマルウェアに奪われてしまう恐れがある。利用アプリはアカウント操作を行うアカウントタイプについて、正規の Authenticator がそのアカウントタイプに割り当てられていることを確認してから、アカウント操作を実施しなければならない。

あるアカウントタイプに割り当てられている Authenticator が正規のものであるかは、その Authenticator を含むパッケージの証明書ハッシュ値を、事前に確認している正規の証明書ハッシュ値と一致するかどうかで確認できる。もし証明書ハッシュ値が一致しないことが判明した場合、そのアカウントタイプに割り当てられている意図しない Authenticator を含むパッケージをアンインストールするようユーザーを促すといった対処を施すことが望ましい。

5.3.3 アドバンスト

5.3.3.1 **Account Manager** の利用と **Permission**

AccountManager クラスの各メソッドを利用するためには、アプリの AndroidManifest.xml にそれぞれ適正な Permission の利用宣言をする必要がある。Android 5.1(API Level 22) 以前のバージョンでは AUTHENTICATE_ACCOUNTS や GET_ACCOUNTS, MANAGE_ACCOUNTS といった権限が必要であり、メソッドとの対応を 表 5.3.1 に示す。

表 5.3.1　Account Manager の機能と Permission

	Account Manager が提供する機能	
Permission	メソッド	説明
AUTHENTICATE_ACCOUNTS (Authenticator と同じ鍵で署名された Package のみ利用可能)	getPassword()	パスワードの取得
	getUserData()	利用者情報の取得
	addAccountExplicitly()	アカウントの DB への追加
	peekAuthToken()	キャッシュされたトークンの取得
	setAuthToken()	認証トークンの登録
	setPassword()	パスワードの変更
	setUserData()	利用者情報の設定
	renameAccount()	アカウント名の変更
GET_ACCOUNTS	getAccounts()	すべてのアカウントの一覧取得
	getAccountsByType()	アカウントタイプが同じアカウントの一覧取得
	getAccountsByTypeAndFeatures()	指定した機能を持ったアカウントの一覧取得
	addOnAccountsUpdatedListener()	イベントリスナーの登録
	hasFeatures()	指定した機能の有無
MANAGE_ACCOUNTS	getAuthTokenByFeatures()	指定した機能を持つアカウントの認証トークンの取得
	addAccount()	ユーザーへのアカウント追加要請
	removeAccount()	アカウントの削除
	clearPassword()	パスワードの初期化
	updateCredentials()	ユーザーへのパスワード変更要請
	editProperties()	Authenticator の設定変更
	confirmCredentials()	ユーザーへのパスワード再入力要請
USE_CREDENTIALS	getAuthToken()	認証トークンの取得
	blockingGetAuthToken()	認証トークンの取得
MANAGE_ACCOUNTS または USE_CREDENTIALS	invalidateAuthToken()	キャッシュされたトークンの削除

ここで、AUTHENTICATE_ACCOUNTS Permission が必要なメソッド群を使う場合には Permission に加えてパッケージの署名鍵に関する制限が設けられている。具体的には、Authenticator を提供するパッケージの署名に使う鍵とメソッドを使うアプリのパッケージの署名に使う鍵が同じでなければならない。そのため、Authenticator 以外に AUTHENTICATE_ACCOUNTS Permission が必要なメソッド群を使うアプリを配布する際には、Authenticator と同じ鍵で署名を施すことになる。

Android 6.0(API Level 23) 以降のバージョンでは GET_ACCOUNTS 以外の Permission は使用されておらず、宣言してもしなくてもできることに差はない。Android 5.1(API Level 22) 以前のバージョンにおいて AUTHENTICATE_ACCOUNTS を要求していたメソッドについては、Permission を要求しないものの、同様に署名が一致する場合のみしか呼び出せない (署名が一致しない場合には SecurityException が発生する) ことに注意する。

さらに、Android 8.0（API Level 26）で GET_ACCOUNTS を必要としていた API のアクセス制御も変更された。Android 8.0（API Level 26）以降のバージョンで、アカウント情報の利用側のアプリの targetSdkVersion が 26 以上の場合には、GET_ACCOUNTS が付与されていたとしても原則として Authenticator アプリと署名が一致する場合にしかアカウントの情報は取得できない。ただし、Authenticator アプリは setAccountVisibility メソッドを呼び出してパッケージ名を指定

することで、署名の一致しないアプリに対してもアカウント情報を提供することができる。

Android Studio での開発の際には設定した署名鍵が固定で使われるため、鍵のことを意識せずに Permission だけで実装や動作確認が出来てしまう。特にアプリによって署名鍵を使い分けている開発者は、この制限を考慮してアプリに使う鍵を選定する必要があるので注意をすること。また、Account Manager から取得するデータにはセンシティブな情報が含まれるため、漏洩や不正利用などのリスクを減らすように扱いには十分注意すること。

5.3.3.2 Android 8.0（API Level 26）以降で署名の一致しない Authenticator のアカウントを読めるケース

Android 8.0（API Level 26）から、アカウント情報の取得など Android 7.1（API Level 25）以前では GET_ACCOUNTS Permission が必要であったメソッドの呼び出しに対して GET_ACCOUNTS Permission が不要になり、代わりに署名が一致する場合や Authenticator アプリ側で setAccountVisibility メソッドによってアカウント情報の提供先アプリとして指定された場合にのみアカウントの情報が取得できるようになった。ただし、フレームワークの課すこのルールにはいくつかの例外があり、注意が必要である。その例外について以下に述べる。

まず、アカウント情報利用側アプリの targetSdkVersion が 25 (Android 7.1) 以下の場合には、上記のルールが適用されず、GET_ACCOUNTS Permission を持っているアプリは署名に関係なく端末内のアカウント情報を取得できる。ただし、この挙動は Authenticator 側の実装によって変更することができることを後に述べる。次に、WRITE_CONTACTS Permission を利用宣言している Authenticator のアカウント情報は、READ_CONTACTS Permission を持っている他アプリから、署名に関係なく読めてしまう。これは、バグではなくフレームワークの仕様である[*11]。ただし、この挙動もまた、Authenticator 側の実装によって変更することができる。

以上のように署名が一致しておらず、かつ setAccountVisibility メソッドの呼び出しによってアカウント情報の提供先に指定していないアプリにもアカウント情報を読まれてしまう例外的なケースはあるのだが、これらの挙動は Authenticator 側で予め次のスニペットのように setAccountVisibility メソッドを呼び出しておくことで変更できる。

第三者アプリにアカウント情報を提供しない

```
accountManager.setAccountVisibility(account, // visibility を変更するアカウント
        AccountManager.PACKAGE_NAME_KEY_LEGACY_VISIBLE,
        AccountManager.VISIBILITY_USER_MANAGED_NOT_VISIBLE);
```

この通りに setAccountVisibility メソッドを呼び出した Authenticator のアカウント情報については、フレームワークはデフォルトの挙動ではなく、targetSdkVersion<=25 のケースや READ_CONTACTS を持っている場合であってもアカウント情報を提供しないように挙動を変更する。

5.4 HTTPS で通信する

スマートフォンアプリはインターネット上の Web サーバーと通信するものが多い。その通信方式として当ガイドでは HTTP と HTTPS の 2 方式に着目する。この 2 方式のうち、セキュリティの観点では HTTPS による通信が望ましい。近年 Google や Facebook など大手の Web サービスを含め、多くのサイトが HTTPS による接続を基本とするように変わってきた。ただし、HTTPS による接続の中でも SSL3.0 や初期の TLS を用いた接続に関しては、POODLE や BEAST 等の脆弱性の存在などによるセキュリティ上の問題が知られており、極力使用しないことを推奨する（「5.4.3.8. （コラム）セキュア接続の *TLS1.2/TLS1.3* への移行について」参照）。

2012 年以降 Android アプリの HTTPS 通信の実装方法における欠陥が多く指摘されている。これは信頼できる第三者認

[*11] WRITE_CONTACTS Permission を利用宣言している Authenticator はアカウント情報を ContactsProvider に書き込むとの想定で、READ_CONTACTS Permission を持つアプリにアカウント情報取得を許可していると考えられる。

証局から発行されたサーバー証明書ではなく、私的に発行されたサーバー証明書（以降、プライベート証明書と呼ぶ）により運用されているテスト用 Web サーバーに接続するために実装された欠陥であると推察される。

この記事では、HTTP および HTTPS 通信の方法について説明する。HTTPS 通信の方法には、プライベート証明書で運用されている Web サーバーに安全に接続する方法も含む。

5.4.1 サンプルコード

開発しているアプリの通信処理の特性を踏まえ、図 5.4.1 に従いサンプルコードを選択すること。

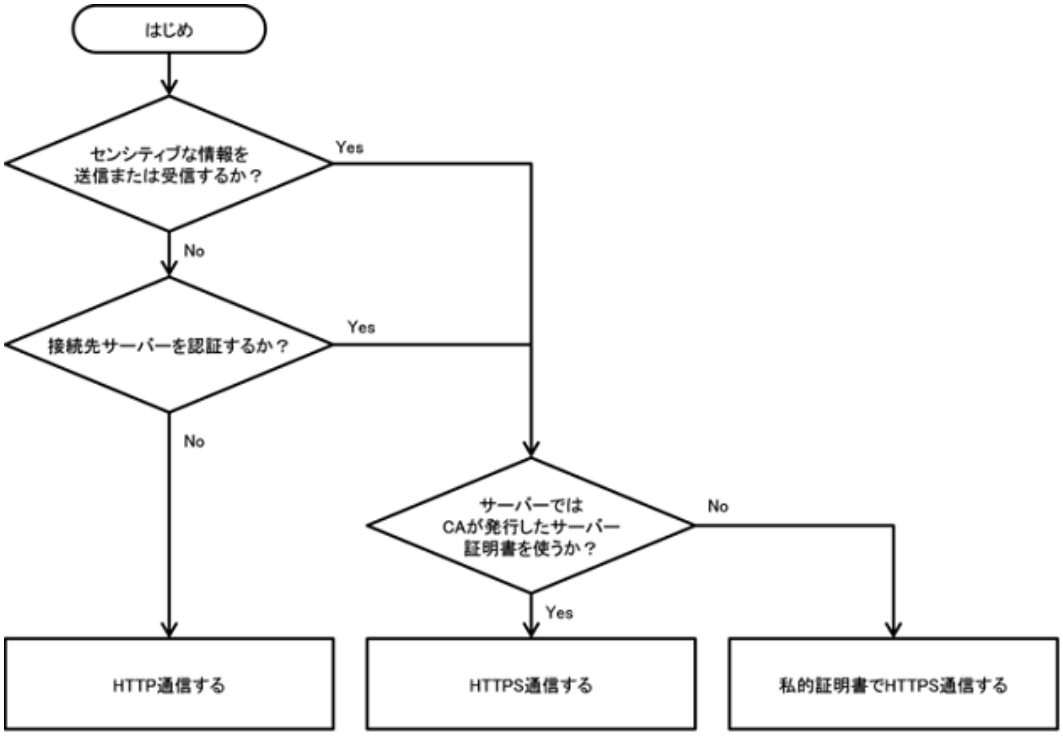

図 5.4.1　HTTP/HTTPS のサンプルコードを選択するフローチャート

センシティブな情報を送受信する場合は SSL/TLS で通信経路が暗号化される HTTPS 通信を用いる。HTTPS 通信を用いたほうが良いセンシティブな情報としては以下のようなものがある。

- Web サービスへのログイン ID、パスワード
- 認証状態を維持するための情報（セッション ID、トークン、Cookie の情報など）
- その他 Web サービスの特性に応じた重要情報・秘密情報（個人情報やクレジットカード情報など）

ここで対象となっているスマートフォンアプリは、サーバーと通信を行うことで連携し、構築されるシステムの一部を担っている。従って、通信のどの部分を HTTP もしくは HTTPS とするのかについては、システム全体を考慮して適切なセキュア設計・セキュアコーディングを施すこと。HTTP と HTTPS の通信方式の違いは 表 5.4.1 を参考にすること。またサンプルコードの違いについては 表 5.4.2 を参考にすること。

表 5.4.1　HTTP 通信方式、HTTPS 通信方式の比較

		HTTP	**HTTPS**
特徴	URL	http://で始まる	https://で始まる
	通信内容の暗号化	なし	あり
	通信内容の改ざん検知	不可	可
	接続先サーバーの認証	不可	可
被害リスク	攻撃者による通信内容の読み取り	高	低
	攻撃者による通信内容の書き換え	高	低
	アプリの偽サーバーへの接続	高	低

表 5.4.2　HTTP/HTTPS 通信のサンプルコードの説明

サンプルコード	通信	センシティブな情報の送受信	サーバー証明書
HTTP 通信する	HTTP	×	-
HTTPS 通信する	HTTPS	o	Cybertrust や VeriSign 等の第三者認証局により発行されたサーバー証明書
プライベート証明書で HTTPS 通信する	HTTPS	o	プライベート証明書※イントラサーバーやテストサーバーで良くみられる運用形態

なお Android がサポートし現在広く使われている HTTP/HTTPS 通信用 API は、Java SDK 由来の java.net.HttpURLConnection/javax.net.ssl.HttpsURLConnection である。Apache HTTPComponent 由来の Apache HttpClient ライブラリについては、Android 6.0（API Level 23) でサポートが打ち切られている。

5.4.1.1 HTTP 通信する

HTTP 通信で送受信する情報はすべて攻撃者に盗聴・改ざんされる可能性があることを前提としなければならない。また、接続先サーバーも攻撃者が用意した偽物のサーバーである可能性がある。このような前提においても被害が生じない、または許容範囲に収まる用途においてのみ、HTTP 通信を利用できる。

以下のサンプルコードは、Web サイト上の画像を HTTP 通信によって取得し表示するアプリである。 非同期処理には Kotlin 標準のコルーチンを使用し、UI スレッドをブロックせずにワーカースレッドで通信処理を行っている。 Web サーバーとの通信で送受信する情報は画像の URL、画像データだが、どれもセンシティブな情報はないとみなしている。そのため、受信データである画像データは攻撃者が用意した攻撃用のデータである可能性がある。簡単のため、サンプルコードでは受信データが攻撃データであるか否かの判定・対策を施していない。同様の理由により、HTTP コネクションおよび画像データを表示するときに発生する可能性のある例外に対する例外処理を最小限にしている。アプリの仕様に応じて適切に処理を実装する必要があることに注意すること。

また、従来の Android では、アプリ全体で HTTP 通信を一律に許可するために AndroidManifest.xml の usesCleartextTraffic 属性を true に設定する手法が用いられていた。 しかし、Android 17 以降、usesCleartextTraffic 属性は非推奨となり、実質的に無効化されることとなった。Android 17 以降の環境では、この属性が true に設定されていても、ネットワークセキュリティ構成による明示的なドメイン許可がない限り、すべての HTTP 通信が OS レベルでブロックされる。 そのため、今後の HTTP 通信の実装では、アプリ一括での通信許可に頼るのではなく、ネットワークセキュリティ構成ファイルを用い、通信が必要な最小限のドメイン（例: www.jssec.org）のみを個別に許可する厳格なアクセス制御が必須となる。

なお、Android 7.0 (API Level 24) 以降では、android:usesCleartextTraffic 属性よりもネットワークセキュリティ構成の設

定が優先されるため、本稿の対応に移行することが強く推奨される。

ポイント：

1. 送信データにセンシティブな情報を含めない
2. 受信データが攻撃者からの送信データである場合を想定する
3. usesCleartextTraffic 属性によるアプリ一括許可は使用せず、ネットワークセキュリティ構成（NSC）を指定する

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" />

    <!-- usesCleartextTraffic="true" は非推奨化のため削除し、networkSecurityConfig を指定 -->
    <application
        android:icon="@mipmap/ic_launcher"
        android:allowBackup="false"
        android:label="@string/app_name"
        android:networkSecurityConfig="@xml/network_security_config" > <!-- ★ポイント 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>
    <!-- デフォルト設定：アプリ全体で非暗号化（HTTP）通信を強制ブロック -->
    <base-config cleartextTrafficPermitted="false" />

    <!-- 1. HTTP 画像検索用（www.jssec.org からの HTTP 取得を例外許可） -->
    <domain-config cleartextTrafficPermitted="true">
        <domain includeSubdomains="true">www.jssec.org</domain>
    </domain-config>
```

(次のページに続く)

(前のページからの続き)

```
    <!-- 2. HTTPS の CT 検証回避テスト用（公開テストドメインを指定） -->
    <domain-config>
        <domain includeSubdomains="true">no-sct.badssl.com</domain>
        <!-- ★ポイント 5★ SCT 情報が含まれていない証明書による通信を一時的に許容するため、CT 検証を無効
化 -->
        <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
 * 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 {

    // ワーカースレッド（Dispatchers.IO）で通信処理を実行する
    suspend fun doInBackground(imageUrl: String): ByteArray? = withContext(Dispatchers.IO)
↪{
        var con: HttpURLConnection? = null
```

(次のページに続く)

(前のページからの続き)

```
        var responseArray: ByteArray? = null
        try {
            // ★ポイント 1 ★ 送信データにセンシティブな情報を含めない
            // 画像 URL の組み立て時などに秘密情報を含めないように注意する
            con = connectUrl(imageUrl) ?: return@withContext null
            checkResponse(con)

            // ★ポイント 2 ★ 受信データが攻撃者からの送信データである場合を想定する
            // サンプルにつき検索結果が攻撃者からのデータ（例：悪意ある画像デコード脆弱性攻撃）である場合の判定・処理は省略
            responseArray = getByteArray(con)
        } catch (e: IOException) {
            // 例外処理は省略
        } 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) {
            // 例外処理は省略
        } catch (e: MalformedURLException) {
            // 例外処理は省略
        } catch (e: IOException) {
            // 例外処理は省略
        }
        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
```

(次のページに続く)

(前のページからの続き)

```
            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) {
                // 例外処理は省略
            }
            try {
                responseArray?.close()
            } catch (e: IOException) {
                // 例外処理は省略
            }
        }
        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
 *
 *      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,
```

(次のページに続く)

(前のページからの続き)

```
 * 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() {
        // Activity が破棄される、またはバックグラウンドへ移行する際に非同期処理をキャンセルしておく
        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)
```

(次のページに続く)

(前のページからの続き)

```
        // 直前の非同期処理が終わっていない可能性を考慮しキャンセル
        mSearchJob?.cancel()

        // lifecycleScope を用いて安全に非同期処理を開始（UI スレッドをブロックしない）
        mSearchJob = lifecycleScope.launch {
            val search = HttpImageSearch()
            val result = search.doInBackground(imageUrl)

            // 結果の反映（自動的に UI スレッド上で安全に実行される）
            if (result == null) {
                mMsgBox.append("\n 例外発生\n")
            } else {
                // サンプルにつき画像デコード・表示の際の例外処理は省略
                val bmp: Bitmap? = BitmapFactory.decodeByteArray(result, 0, result.size)
                mImgBox.setImageBitmap(bmp)
            }
        }
    }

    fun onHttpsSearchClick(view: View) {
        val query = mQueryBox.text.toString()
        mMsgBox.text = "HTTPS 検索開始: $query"
        mImgBox.setImageBitmap(null)

        // 前回の検索タスクが動作中であればキャンセルする
        mSearchJob?.cancel()

        // lifecycleScope を用いて安全に非同期 HTTPS 通信を開始
        mSearchJob = lifecycleScope.launch {
            val search = HttpsImageSearch()
            val result = search.doInBackground(query)

            // 結果を UI に反映
            if (result == null) {
                mMsgBox.append("\n 例外発生（通信失敗）\n")
            } else {
                // 画像データのデコード例外処理はアプリ要件に応じて適切に行うこと
                val bmp: Bitmap? = BitmapFactory.decodeByteArray(result, 0, result.size)
                mImgBox.setImageBitmap(bmp)
                mMsgBox.text = "HTTPS 検索完了: $query"
            }
        }
    }
}
```

5.4.1.2 HTTPS 通信する

HTTPS 通信では送受信するデータが暗号化されるだけでなく、接続先サーバーが本物かどうかの検証も行われる。そのために HTTPS 通信開始時のハンドシェイク処理において、サーバーから送られてくるサーバー証明書に対して、Android の HTTPS ライブラリ内部で次のような観点で検証が行われる。

- 第三者認証局により署名されたサーバー証明書であること
- サーバー証明書の期限等が有効であること
- サーバー証明書の Subject の CN（Common Name）または SAN（Subject Alternative Names）の DNS 名が接続先サーバーのホスト名と一致していること

これらの検証に失敗するとサーバー証明書検証エラーが発生する。サーバー証明書に不備がある場合、もしくは攻撃者が中間者攻撃をしている場合にこのエラーが発生する。エラーが発生した場合には、アプリの仕様に応じて適切な処理を実行する必要がある点に注意すること。

Android 17 以上をターゲットとするアプリでは、HTTPS 通信の安全性をさらに高めるため、以下の強固なセキュリティ機能が OS レベルでデフォルト有効化される。

1. ECH（Encrypted Client Hello）のデフォルト有効化

従来の TLS（HTTPS）通信では、通信中身自体は暗号化されていたものの、ハンドシェイク時に接続先ホスト名を提示する「SNI（Server Name Indication）」が平文で送信されていたため、通信経路上の第三者にアクセス先ドメイン名（＝利用しているサービス）を覗き見られるリスクが存在していた。Android 17 以降では、この SNI 部分も暗号化する「ECH」がデフォルトで有効化される。これにより、ネットワーク監視者に接続先の特定ドメインを把握されるリスクが極めて低くなり、通信プライバシーが大幅に強化される。 ※ ECH を実際に機能させるには、アプリの通信ライブラリおよび、接続先サーバー/CDN が ECH プロトコルをサポートしている必要がある。

2. 証明書の透明性（Certificate Transparency: CT）のデフォルト検証

TLS 証明書の誤発行や不正発行による中間者攻撃を防ぐため、Android 17 以降では公開信頼された TLS 証明書に対する CT 検証がデフォルトで自動適用される。これにより、接続先証明書が公開ログ（CT ログ）に記録し、ログ登録証明として SCT（Signed Certificate Timestamp）を付与、クライアント（Android など）はこの SCT を検証する。つまり、証明書が公開ログに載っているかをチェックして不正な証明書を検出する仕組みである。 SCT が含まれていない（CT 非対応の）証明書を使用しているサーバーに接続しようとすると、ハンドシェイク時に SSLHandshakeException が発生して通信が強制遮断される。 ※本番環境の公的 CA（Let’s Encrypt や DigiCert など）から取得した証明書は問題なく CT をパスするが、開発環境やステージング環境で自己署名証明書を使用する場合は、CT ログに登録されないため接続エラーになる。この場合は、後述の通りネットワークセキュリティ構成を用いて、対象ドメインのみ例外的に CT 検証を無効化（enabled=”false”）する必要がある。

ポイント：

1. URI は https://で始める
2. 送信データにセンシティブな情報を含めてよい
3. HTTPS 接続したサーバーからのデータであっても、受信データの安全性を確認する
4. SSLException に対してアプリに適した例外処理を行う
5. 開発・テスト環境で SCT のない証明書や自己署名証明書を使う場合は、ネットワークセキュリティ構成で CT 検証を無効化（自己署名の場合はさらにプライベート CA の信頼設定を追加）する

network_security_config.xml

```
<?xml version="1.0" encoding="utf-8"?>
<network-security-config>
    <!-- デフォルト設定：アプリ全体で非暗号化（HTTP）通信を強制ブロック -->
    <base-config cleartextTrafficPermitted="false" />

    <!-- 1. HTTP 画像検索用（www.jssec.org からの HTTP 取得を例外許可） -->
    <domain-config cleartextTrafficPermitted="true">
        <domain includeSubdomains="true">www.jssec.org</domain>
    </domain-config>

    <!-- 2. HTTPS の CT 検証回避テスト用（公開テストドメインを指定） -->
    <domain-config>
        <domain includeSubdomains="true">no-sct.badssl.com</domain>
        <!-- ★ポイント 5★ SCT 情報が含まれていない証明書による通信を一時的に許容するため、CT 検証を無効
化 -->
        <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
```

(次のページに続く)

(前のページからの続き)

```
import java.net.ProtocolException
import java.net.URL
import javax.net.ssl.HttpsURLConnection
import javax.net.ssl.SSLException

class HttpsImageSearch {

    // ワーカースレッド（Dispatchers.IO）で安全に HTTPS 通信を実行する
    suspend fun doInBackground(query: String): ByteArray? = withContext(Dispatchers.IO) {
        var con: HttpsURLConnection? = null
        var responseArray: ByteArray? = null
        try {
            // 1 回目の通信：画像の検索を実行
            // ★ポイント 1★ URI は https://で始める
            // ★ポイント 2★ 暗号化されるため送信データにパスワードやトークン等のセンシティブ情報を含めてもよい
            val searchUrl = "https://www.jssec.org/"
            con = connectUrl(searchUrl) ?: return@withContext null
            checkResponse(con)

            // 検索結果から画像の URL を取得する処理（サンプルにつきモック URL を使用）
            val imageUrl = "https://www.jssec.org/common/images/main_visual_local.jpg"
            con.disconnect()

            // 2 回目の通信：画像データを取得
            con = connectUrl(imageUrl) ?: return@withContext null
            checkResponse(con)

            // ★ポイント 3★ HTTPS 経由であっても、受信データ（画像）の安全性はデコード時に確認する
            responseArray = getByteArray(con)

        } catch (e: SSLException) {
            // ★ポイント 4★ 証明書検証エラー（中間者攻撃や期限切れなど）発生時の例外処理
            // 実商用アプリでは、ユーザーへエラーダイアログ等で警告を促す処理などを適切に実装すること
            e.printStackTrace()
        } catch (e: IOException) {
            // その他の通信例外処理
            e.printStackTrace()
        } finally {
            con?.disconnect()
        }
        return@withContext responseArray
    }

    private fun connectUrl(strUrl: String): HttpsURLConnection? {
```

(次のページに続く)

(前のページからの続き)

```
    var con: HttpsURLConnection? = null
    try {
        val url = URL(strUrl)
        con = url.openConnection() as HttpsURLConnection
        con.requestMethod = "GET"
        con.connect()
    } catch (e: ProtocolException) {
        // 例外処理は省略
    } catch (e: MalformedURLException) {
        // 例外処理は省略
    } catch (e: IOException) {
        // 例外処理は省略
    }
    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) {
                responseArray.write(buff, 0, length)
            }
        }
        result = responseArray.toByteArray()
    } catch (e: IOException) {
        e.printStackTrace()
    } finally {
        try {
            inputStream?.close()
        } catch (e: IOException) {
            // 例外処理は省略
        }
        try {
            responseArray?.close()
        } catch (e: IOException) {
            // 例外処理は省略
        }
    }
```

(次のページに続く)

(前のページからの続き)

```
        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 プライベート証明書で HTTPS 通信する

ここでは第三者認証局から発行されたサーバー証明書ではなく、私的に発行したサーバー証明書（プライベート証明書）で HTTPS 通信をするサンプルコードを示す。プライベート認証局のルート証明書とプライベート証明書の作成方法および Web サーバーの HTTPS 設定については「5.4.3.1. プライベート証明書の作成方法とサーバー設定」を参考にすること。またサンプルプログラムの assets 中の cacert.crt ファイルはプライベート認証局のルート証明書ファイルである。

以下のサンプルコードは、Web サーバー上の画像を取得して表示するアプリである。Web サーバーとは HTTPS を用いた通信を行う。UI スレッドでの通信を避けるために、AsyncTask を利用して通信処理用のワーカースレッドを作成している。Web サーバーとの通信で送受信する情報は画像の URL と画像データで、このサンプルではどちらもセンシティブな情報とみなしている。また、簡単のため、SSLException に対してはユーザーへの通知などの例外処理を行っていないが、アプリの仕様に応じて適切な処理を実装する必要がある。

ポイント：

1. プライベート認証局のルート証明書でサーバー証明書を検証する
2. URI は https://で始める
3. 送信データにセンシティブな情報を含めてよい
4. 受信データを接続先サーバーと同じ程度に信用してよい
5. SSLException に対しユーザーに通知する等の適切な例外処理をする

PrivateCertificateHttpsGet.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
```

(次のページに続く)

(前のページからの続き)

```
 * 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]);
            // ★ポイント 1★ プライベート認証局のルート証明書でサーバー証明書を検証する
```

(次のページに続く)

(前のページからの続き)

```
            // assets に格納しておいたプライベート証明書だけを含む KeyStore を設定
            KeyStore ks = KeyStoreUtil.getEmptyKeyStore();
            KeyStoreUtil.loadX509Certificate(ks,
                mContext.getResources().getAssets().open("cacert.crt"));

            // ホスト名の検証を行う
            HttpsURLConnection.setDefaultHostnameVerifier(new HostnameVerifier() {
                @Override
                public boolean verify(String hostname, SSLSession session) {
                    if (!hostname.equals(session.getPeerHost())) {
                        return false;
                    }
                    return true;
                }
            });

            // ★ポイント 2★ URI は https://で始める
            // ★ポイント 3★ 送信データにセンシティブな情報を含めてよい
            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);

            // ★ポイント 4★ 受信データを接続先サーバーと同じ程度に信用してよい
            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) {
            // ★ポイント 5★ SSLException に対しユーザーに通知する等の適切な例外処理をする
            // サンプルにつき例外処理は割愛
            return e;
        } catch(Exception e) {
```

(次のページに続く)

(前のページからの続き)

```
            return e;
        } finally {
            if (inputStream != null) {
                try {
                    inputStream.close();
                } catch (Exception e) {
                    // 例外処理は割愛
                }
            }
            if (responseArray != null) {
                try {
                    responseArray.close();
                } catch (Exception e) {
                    // 例外処理は割愛
                }
            }
        }
    }

    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,
```

(次のページに続く)

(前のページからの続き)

```
                                                    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)
        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) { /* 例外処理は割愛 */ }
        }
    }
}
```

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
 *
```

(次のページに続く)

(前のページからの続き)

```
 * 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);
        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() {
        // このあと Activity が破棄される可能性があるので非同期処理をキャンセルしておく
        if (mAsyncTask != null) mAsyncTask.cancel(true);
        super.onPause();
    }

    public void onClick(View view) {
        String url = mUrlBox.getText().toString();
```

(次のページに続く)

（前のページからの続き）

```
        mMsgBox.setText(url);
        mImgBox.setImageBitmap(null);

        // 直前の非同期処理が終わってないこともあるのでキャンセルしておく
        if (mAsyncTask != null) mAsyncTask.cancel(true);

        // UI スレッドで通信してはならないので、AsyncTask によりワーカースレッドで通信する
        mAsyncTask = new PrivateCertificateHttpsGet(this) {
                @Override
                protected void onPostExecute(Object result) {
                    // UI スレッドで通信結果を処理する
                    if (result instanceof Exception) {
                        Exception e = (Exception)result;
                        mMsgBox.append("\n 例外発生\n" + e.toString());
                    } else {
                        byte[] data = (byte[])result;
                        Bitmap bmp = BitmapFactory.decodeByteArray(data, 0, data.length);
                        mImgBox.setImageBitmap(bmp);
                    }
                }
            }.execute(url);      // URL を渡して非同期処理を開始
    }
}
```

5.4.2 ルールブック

HTTP 通信、HTTPS 通信する場合には以下のルールを守ること。

1. センシティブな情報は *HTTPS* 通信で送受信する （必須）
2. *HTTP* 通信では受信データの安全性を確認する （必須）
3. *SSLException* に対しユーザーに通知する等の適切な例外処理をする （必須）
4. 独自の *TrustManager* を作らない （必須）
5. 独自の *HostnameVerifier* は作らない （必須）
6. *HTTP* 通信を行う場合は、*usesCleartextTraffic* 属性による一括許可は使用せず、ネットワークセキュリティ構成で必要なドメインのみを個別に許可する （必須）
7. 開発・テスト環境で *SCT* 情報がない証明書や自己署名証明書を使う場合は、ネットワークセキュリティ構成で *CT* 検証を安全にオプトアウトする （必須）

5.4.2.1 センシティブな情報は **HTTPS** 通信で送受信する （必須）

HTTP を使った通信では、送受信する情報の盗聴・改ざん、または、接続先サーバーのなりすましが起こる可能性がある。センシティブな情報は HTTPS 通信で送受信すること。

5.4.2.2 **HTTP** 通信では受信データの安全性を確認する （必須）

HTTP 通信における受信データは攻撃者が制御可能であるため、コード脆弱性を狙った攻撃データを受信する可能性がある。あらゆる値、形式のデータを受信することを想定して、受信データを処理するコードに脆弱性がないように気を付けてコーディングする必要がある。また、HTTPS 通信における受信データについても、受信データを無条件に安全であると考えてはならない。HTTPS 接続先のサーバーが攻撃者によって用意されたものである場合や、受信データが接続先サーバーとは別の場所で生成されたデータである場合もあるためである。「3.2. 入力データの安全性を確認する」も参照すること。

5.4.2.3 **SSLException** に対しユーザーに通知する等の適切な例外処理をする （必須）

HTTPS 通信ではサーバー証明書の検証時に SSLException が発生することがある。SSLException はサーバー証明書の不備が原因となって発生する。証明書の不備は攻撃者による中間者攻撃によって発生している可能性があるので、SSLException に対しては適切な例外処理を実装することが必要である。例外処理の例としては、SSLException による通信失敗をユーザーに通知すること、あるいはログに記録することが考えられる。その一方で、アプリによってはユーザーに対する特別な通知は必要とされないこともありうる。このように、実装すべき処理はアプリの仕様や特性によって異なるので、それらを十分に検討した上で決定しなければならない。

加えて、SSLException が発生した場合には中間者攻撃を受けている可能性があるので、HTTP などの非暗号化通信によってセンシティブな情報の送受信を再度試みるような実装してはならない。

5.4.2.4 独自の **TrustManager** を作らない （必須）

自己署名証明書などのプライベート証明書で HTTPS 通信するためには、サーバー証明書検証に使う KeyStore を変更するだけで済む。しかしながら「5.4.3.3. 証明書検証を無効化する危険なコード」で説明しているように、インターネット上で公開されているサンプルコードには、危険な TrustManager を実装する例を紹介しているものが多くある。これらのサンプルを参考にして実装されたアプリは、脆弱性を作りこむ可能性がある。

プライベート証明書で HTTPS 通信をしたい場合には「5.4.1.3. プライベート証明書で *HTTPS* 通信する」の安全なサンプルコードを参照すること。

本来ならば独自の TrustManager を安全に実装することも可能であるが、暗号処理や暗号通信に十分な知識をもった技術者でなければミスを作り込む危険性があるため、このルールはあえて必須とした。

5.4.2.5 独自の **HostnameVerifier** は作らない （必須）

自己署名証明書などのプライベート証明書で HTTPS 通信するためには、サーバー証明書検証に使う KeyStore を変更するだけで済む。しかしながら「5.4.3.3. 証明書検証を無効化する危険なコード」で説明しているように、インターネット上で公開されているサンプルコードには、危険な HostnameVerifier を利用する例を紹介しているものが多くある。これらのサンプルを参考にして実装したアプリは、脆弱性を作りこむ可能性がある。

プライベート証明書で HTTPS 通信をしたい場合には「5.4.1.3. プライベート証明書で *HTTPS* 通信する」の安全なサンプルコードを参照すること。

本来ならば独自の HostnameVerifier を安全に実装することも可能であるが、暗号処理や暗号通信に十分な知識をもった技術者でなければミスを作り込む危険性があるため、このルールはあえて必須とした。

5.4.2.6 HTTP 通信を行う場合は、usesCleartextTraffic 属性による一括許可は使用せず、ネットワークセキュリティ構成で必要なドメインのみを個別に許可する （必須）

Android 17 以降、AndroidManifest.xml の usesCleartextTraffic 属性は非推奨となり、実質的に無効化される。

この属性を true に設定していても、ネットワークセキュリティ構成ファイルによる明示的なドメイン許可がない限り、すべての HTTP 通信が OS レベルでブロックされる。

そのため、HTTP 通信が必要な場合は、アプリ一括許可に頼るのではなく、ネットワークセキュリティ構成ファイルを用いて、やむを得ずクリアテキスト通信を行う最小限のドメインのみを個別に許可（cleartextTrafficPermitted=”true”）しなければならない。

5.4.2.7 開発・テスト環境で SCT 情報がない証明書や自己署名証明書を使う場合は、ネットワークセキュリティ構成で CT 検証を安全にオプトアウトする （必須）

Android 17 以降をターゲットとするアプリでは、HTTPS 通信時に「証明書の透明性（CT）」の自動検証がデフォルトで有効化される。

本番環境の公的 CA（Let’s Encrypt 等）から取得した証明書は問題なくパスするが、検証用に構築したステージングサーバーなど、SCT 情報が含まれていない証明書や自己署名証明書を使用しているサーバーに接続しようとすると、SSLHandshakeException が発生して通信が強制遮断される。

このため、開発・テスト用の特定の通信先に限っては、ネットワークセキュリティ構成において対象ドメインを指定し、例外的に CT 検証を無効化（<certificateTransparency enabled=”false” />）する必要がある。

なお、自己署名証明書を使用する場合は、CT 検証の無効化だけでなく、そのプライベート CA 証明書自体をアプリに信頼させる設定（<trust-anchors>）を同時に記述しなければならない点に注意すること。

5.4.3 アドバンスト

5.4.3.1 プライベート証明書の作成方法とサーバー設定

ここでは Ubuntu や CentOS などの Linux 環境におけるプライベート証明書の作成方法とサーバー設定について説明する。プライベート証明書は私的に発行されたサーバー証明書のことである。Cybertrust や VeriSign などの第三者認証局から発行されたサーバー証明書と区別してプライベート証明書と呼ばれる。

プライベート認証局の作成

まずプライベート証明書を発行するためのプライベート認証局を作成する。Cybertrust や VeriSign などの第三者認証局と区別してプライベート認証局と呼ばれる。1 つのプライベート認証局で複数のプライベート証明書を発行できる。プライベート認証局を作成した PC は、限られた信頼できる人物しかアクセスできないように厳重に管理されなければならない。

プライベート認証局を作成するには、下記のシェルスクリプト newca.sh および設定ファイル openssl.cnf を作成し実行する。シェルスクリプト中の CASTART および CAEND は認証局の有効期間、CASUBJ は認証局の名称であるので、作成する認証局に合わせて変更すること。シェルスクリプト実行の際には認証局アクセスのためのパスワードが合計 3 回聞かれるので、同じパスワードを入力すること。

newca.sh -- プライベート認証局を作成するシェルスクリプト

```
#!/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 - 2 つのシェルスクリプトが共通に参照する openssl コマンドの設定ファイル

```
[ 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
```

(次のページに続く)

(前のページからの続き)

```
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

[ 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}
```

上記シェルスクリプトを実行すると、作業ディレクトリ直下に CA というディレクトリが作成される。この CA ディレクトリがプライベート認証局である。CA/cacert.crt ファイルがプライベート認証局のルート証明書であり、「5.4.1.3. プライベート証明書で *HTTPS* 通信する」の assets に使用されたり、「5.4.3.2. *Android OS* の証明書ストアにプライベート認証局のルート証明書をインストールする」で Android 端末にインストールされたりする。

プライベート証明書の作成

プライベート証明書を作成するには、下記のシェルスクリプト newsv.sh を作成し実行する。シェルスクリプト中の SVSTART および SVEND はプライベート証明書の有効期間、SVSUBJ は Web サーバーの名称であるので、対象 Web サーバーに合わせて変更すること。特に SVSUBJ の/CN で指定する Web サーバーのホスト名はタイプミスがないように気を付けること。シェルスクリプトを実行すると認証局アクセスのためのパスワードが聞かれるので、プライベート認証局を作成するときに指定したパスワードを入力すること。その後、合計 2 回の y/n を聞かれるので y を入力すること。

newsv.sh - プライベート証明書を発行するシェルスクリプト

```
#!/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"

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}
```

上記シェルスクリプトを実行すると、作業ディレクトリ直下に Web サーバー用のプライベートキーファイル svkey.pem およびプライベート証明書ファイル svcert.pem が生成される。

Web サーバーが Apache である場合には、設定ファイル中に上で作成した prikey.pem と cert.pem を次のように指定するとよい。

```
SSLCertificateFile "/path/to/svcert.pem"
SSLCertificateKeyFile "/path/to/svkey.pem"
```

5.4.3.2 Android OS の証明書ストアにプライベート認証局のルート証明書をインストールする

「5.4.1.3. プライベート証明書で *HTTPS* 通信する」のサンプルコードは、1 つのアプリにプライベート認証局のルート証明書を持たせることで、プライベート証明書で運用する Web サーバーに HTTPS 接続する方法を紹介した。ここでは Android OS にプライベート認証局のルート証明書をインストールすることで、すべてのアプリがプライベート証明書で運用する Web サーバーに HTTPS 接続する方法を紹介する。インストールしてよいのは、信頼できる認証局の発行した証明書に限ることに注意すること。

ただし、ここで述べる方法は Android 6.0(API Level 23) 以前のバージョンにのみ適用可能なものである。Android 7.0(API Level 24) 以降では、プライベート認証局のルート証明書をインストールしても、システムはこれを無視する。API Level 24 以降でプライベート証明書を利用したい場合は、「5.4.3.7. *Network Security Configuration*」の「プライベート証明書で HTTPS 通信する」の項を参照されたい。

まずプライベート認証局のルート証明書ファイル cacert.crt を Android 端末の内部ストレージにコピーする。なおサンプルコードで使用しているルート証明書ファイルは https://www.jssec.org/dl/android_securecoding_sample_cacert.crt からも取得できる。

次に Android の設定メニューのセキュリティを開き、下図のような手順を進めることで Android OS にルート証明書をインストールすることができる。

図 5.4.2　プライベート認証局のルート証明書のインストール手順

図 5.4.3　ルート証明書がインストールされていることの確認

Android OS にプライベート認証局のルート証明書をインストールすると、その認証局から発行されたプライベート証明書をすべてのアプリで正しく証明書検証できるようになる。下図は Chrome ブラウザで https://selfsigned.jssec.org/droid_knight.png を表示した場合の例である。

図 5.4.4　ルート証明書のインストール後はプライベート証明書を正しく検証できるようになる

この方法を使えば「5.4.1.2. *HTTPS* 通信する」のサンプルコードでもプライベート証明書で運用する Web サーバーに HTTPS 接続できるようになる。

5.4.3.3 証明書検証を無効化する危険なコード

インターネット上にはサーバー証明書検証エラーを無視して HTTPS 通信をするサンプルコードが多数掲載されている。これらのサンプルコードはプライベート証明書を使って HTTPS 通信を実現する方法として紹介されているため、そうしたサンプルコードをコピー＆ペーストして利用しているアプリが多数存在している。残念ながらこうしたサンプルコードは中間者攻撃に脆弱なものであることが多く、この記事の冒頭で「2012 年には Android アプリの HTTPS 通信の実装方法における欠陥が多く指摘された。」と述べたように、こうしたインターネット上の脆弱なサンプルコードを利用してしまったと思われる多くの脆弱な Android アプリが報告されている。

ここではこうした脆弱な HTTPS 通信のサンプルコードの断片を紹介する。こうしたサンプルコードを見かけた場合には「5.4.1.3. プライベート証明書で *HTTPS* 通信する」のサンプルコードに置き換えるなどしていただきたい。

危険：空っぽの TrustManager を作るケース

```
TrustManager tm = new X509TrustManager() {

    @Override
    public void checkClientTrusted(X509Certificate[] chain,
                                   String authType) throws CertificateException {
```

(次のページに続く)

(前のページからの続き)

```
        // 何もしない → どんな証明書でも受付ける
    }

    @Override
    public void checkServerTrusted(X509Certificate[] chain,
                                   String authType) throws CertificateException {
        // 何もしない → どんな証明書でも受付ける
    }

    @Override
    public X509Certificate[] getAcceptedIssuers() {
        return null;
    }
};
```

危険：空っぽの HostnameVerifier を作るケース

```
HostnameVerifier hv = new HostnameVerifier() {
    @Override
    public boolean verify(String hostname, SSLSession session) {
        // 常に true を返す → どんなホスト名でも受付ける
        return true;
    }
};
```

危険：ALLOW_ALL_HOSTNAME_VERIFIER を使っているケース

```
SSLSocketFactory sf;
// …
sf.setHostnameVerifier(SSLSocketFactory.ALLOW_ALL_HOSTNAME_VERIFIER);
```

5.4.3.4 HTTP リクエストヘッダを設定する際の注意点

HTTP および HTTPS 通信において、独自の HTTP リクエストヘッダを設定したい場合は、URLConnection クラスの setRequestProperty() メソッド、もしくは addRequestProperty() メソッドを使用する。これらメソッドの引数に外部からの入力データを用いる場合は、HTTP ヘッダ・インジェクションの対策が必要となる。HTTP ヘッダ・インジェクションによる攻撃の最初のステップとなるのは、HTTP ヘッダの区切り文字である改行コードを入力データに含めることであるため、入力データから改行コードを排除するようにしなければならない。

HTTP リクエストヘッダを設定する

```
public byte[] openConnection(String strUrl, String strLanguage, String strCookie) {
    // HttpURLConnection は URLConnection の派生クラス
    HttpURLConnection connection;
```

(次のページに続く)

(前のページからの続き)

```
try {
    URL url = new URL(strUrl);
    connection = (HttpURLConnection) url.openConnection();
    connection.setRequestMethod("GET");

    // ★ポイント★ HTTP リクエストヘッダに入力値を使用する場合は、アプリケーション要件に従って
    // 入力データをチェックする (※)
    if (strLanguage.matches("^[a-zA-Z ,-]+$")) {
        connection.addRequestProperty("Accept-Language", strLanguage);
    } else {
        throw new IllegalArgumentException("Invalid Language : " + strLanguage);
    }
    // ★ポイント★ もしくは入力データを URL エンコードする (というアプリケーション要件にする)
    connection.setRequestProperty("Cookie", URLEncoder.encode(strCookie, "UTF-8"));

    connection.connect();

    // ～省略～
```

※「3.2. 入力データの安全性を確認する」を参照

5.4.3.5 ピンニングによる検証の注意点と実装例

アプリが HTTPS 通信を行う際は、通信開始時のハンドシェイク処理において、接続先サーバーから送られてくる証明書が第三者認証局により署名されているかどうかの検証が行われる。しかし、攻撃者が第三者認証局から不正な証明書を入手したり、認証局の署名鍵を入手して不正な証明書を作成したりした場合、その攻撃者により不正なサーバーへの誘導や中間者攻撃が行われても、アプリはそれらの攻撃をハンドシェイク処理で検出することができず、結果として被害につながってしまう可能性がある。

このような不正な第三者認証局の証明書を用いた中間者攻撃に対しては「ピンニングによる検証」が有効である。これは、あらかじめ接続先サーバーの証明書や公開鍵をアプリ内に保持しておき、それらの情報をハンドシェイク処理で用いたり、ハンドシェイク処理後に再検証したりする方法である。

ピンニングによる検証は、公開鍵基盤（PKI）の基礎である第三者認証局の信頼性が損なわれた場合に備え、通信の安全性を補填する目的で用いられる。開発者は自身のアプリが扱う資産レベルに応じて、この検証を行うかどうか検討してほしい。

アプリ内に保持した証明書・公開鍵をハンドシェイク処理で使用する

アプリ内に保持しておいた接続先サーバーの証明書や公開鍵の情報をハンドシェイク処理で用いるためには、それらの情報を含めた独自の KeyStore を作成して通信に用いる。これにより、上記のような不正な第三者認証局の証明書を用いた中間者攻撃が行われても、ハンドシェイク処理において不正を検出することができるようになる。独自の KeyStore を設定して HTTPS 通信を行う具体的な方法は「5.4.1.3. プライベート証明書で *HTTPS* 通信する」で紹介したサンプルコードを参照すること。

アプリ内に保持した証明書・公開鍵を用いてハンドシェイク処理後に再検証する

ハンドシェイク処理が行われた後に接続先を再検証するためには、まずハンドシェイク処理で検証されシステムに信頼

された証明書チェーンを取得し、その証明書チェーンを、あらかじめアプリ内に保持しておいた情報と照合する。照合の結果、保持しておいた情報と一致するものが含まれていれば通信を許可し、含まれていなければ通信処理を中断させればよい。ただし、ハンドシェイク処理でシステムに信頼された証明書チェーンを取得する際に以下のメソッドを使用すると、期待通りの証明書チェーンが得られず、結果としてピンニングによる検証が正常に機能しなくなってしまう危険がある[*12]。

- javax.net.ssl.SSLSession.getPeerCertificates()
- javax.net.ssl.SSLSession.getPeerCertificateChain()

これらのメソッドが返すのは、ハンドシェイク処理でシステムに信頼された証明書チェーンではなく、アプリが通信相手から受け取った証明書チェーンそのものである。そのため、中間者攻撃により不正な第三者認証局の証明書が証明書チェーンに付け加えられても、上記のメソッドはハンドシェイク処理でシステムが信用した証明書だけでなく、本来アプリが接続しようとしていたサーバーの証明書も一緒に返してしまう。この「本来アプリが接続しようとしていたサーバーの証明書」は、ピンニングによる検証のためアプリ内にあらかじめ保持しておいたものと同等の証明書なので、再検証を行っても不正を検出することができない。このような理由から、ハンドシェイク処理後の再検証を実装する際に上記のメソッドを使用することは避けるべきである。

Android 4.2（API Level 17）以上であれば、上記のメソッドの代わりに net.http.X509TrustManagerExtensions の checkServerTrusted() を使用することで、ハンドシェイク処理でシステムに信頼された証明書チェーンのみを取得することができる。

X509TrustManagerExtensions を用いたピンニング検証の例

```
// 正しい通信先サーバーの証明書に含まれる公開鍵の SHA-256 ハッシュ値を保持（ピンニング）
private static final Set<String> PINS = new HashSet<>(Arrays.asList(
    new String[] {
        "d9b1a68fceaa460ac492fb8452ce13bd8c78c6013f989b76f186b1cbba1315c1",
        "cd13bb83c426551c67fabcff38d4496e094d50a20c7c15e886c151deb8531cdc"
    }
));

// AsyncTask のワーカースレッドで通信する
protected Object doInBackground(String... strings) {

    // ～省略～

    // ハンドシェイク時の検証によりシステムに信頼された証明書チェーンを取得する
    X509Certificate[] chain = (X509Certificate[]) connection.getServerCertificates();
    X509TrustManagerExtensions trustManagerExt =
        new X509TrustManagerExtensions((X509TrustManager) (trustManagerFactory
                                                            .getTrustManagers()[0]));
    List<X509Certificate> trustedChain =
        trustManagerExt.checkServerTrusted(chain, "RSA", url.getHost());

    // 公開鍵ピンニングを用いて検証する
```

（次のページに続く）

[*12] この危険性については、https://www.synopsys.com/blogs/software-security/ineffective-certificate-pinning-implementations/ の記事で詳しく説明されている。

(前のページからの続き)

```
    boolean isValidChain = false;
    for (X509Certificate cert : trustedChain) {
        PublicKey key = cert.getPublicKey();
        MessageDigest md = MessageDigest.getInstance("SHA-256");
        String keyHash = bytesToHex(md.digest(key.getEncoded()));

        // ピンニングしておいた公開鍵のハッシュ値と比較する
        if(PINS.contains(keyHash)) isValidChain = true;
    }
    if (isValidChain) {
        // 処理を継続する
    } else {
        // 処理を継続しない
    }

    // ～省略～
}

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();
}
```

5.4.3.6 Google Play 開発者サービスを利用した OpenSSL の脆弱性対策

Google Play 開発者サービス（バージョン 5.0 以降）では、Provider Installer という仕組みが提供されている。これは、OpenSSL を含む暗号関連技術の実装である Security Provider の脆弱性対策に利用できる。詳しくは「5.6.3.4. *Google Play 開発者サービスによる Security Provider の脆弱性対策*」を参照のこと。

5.4.3.7 Network Security Configuration

Android 7.0（API Level 24）において、ネットワーク通信時のセキュリティ設定をアプリ毎に行うことができる Network Security Configuration が導入された。この仕組みを利用することにより、プライベート証明書での HTTPS 通信やピンニングによる証明書検証のほか、非暗号化（HTTP）通信の抑制、デバッグ時のみ有効なプライベート証明書の導入など、アプリのセキュリティを向上させる種々の施策をアプリに簡単に取り入れることができる[*13]。

Network Security Configuration の各種機能は xml ファイルの設定を行うだけで使用でき、アプリが行う HTTP および HTTPS 通信全てに適用することができる。その結果、アプリのコードに修正や追加処理を行う必要がなくなるため、実装がシンプルになりバグや脆弱性の作り込み防止に効果があると考えられる。

[*13] Network Security Configuration の詳細については、https://developer.android.com/training/articles/security-config.html を参照すること。

プライベート証明書で **HTTPS** 通信する

「5.4.1.3. プライベート証明書で *HTTPS* 通信する」で、私的に発行したサーバー証明書（プライベート証明書）で HTTPS 通信をするためのサンプルコードを示した。Network Security Configuration を用いれば、開発者がプライベート証明書の検証処理を明示的に実装しなくても、「5.4.1.2. *HTTPS* 通信する」のサンプルコードで示した通常の HTTPS 通信と同じ実装でプライベート証明書を用いることができる。

特定ドメインへの通信時にプライベート証明書を用いる

```
<?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>
```

上記の例では、通信で使用するプライベート証明書（private_ca）をアプリ内にリソースとして保持しておき、それらを利用する条件や適用範囲を xml ファイルに記述している。<domain-config>タグを使用することで特定のドメインに対してのみプライベート証明書が適用される。アプリが行う全ての HTTPS 通信に対してプライベート証明書を用いるためには、以下のように<base-config>タグを用いればよい。

アプリが行う全ての HTTPS 通信時にプライベート証明書を用いる

```
<?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>
```

ピンニングによる検証

「5.4.3.5. ピンニングによる検証の注意点と実装例」でピンニングによる証明書の検証について説明した。Network Security Configuration を用い以下のように設定すれば、コード上の検証処理が不要となり、xml の記述だけで検証を行うことができる。

HTTPS 通信時にピンニングによる検証を行う

```
<?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>
```

(次のページに続く)

(前のページからの続き)

```
            <pin digest="SHA-256">fwza0LRMXouZHRC8Ei+4PyuldPDcf3UKgO/04cDM1oE=</pin>
        </pin-set>
    </domain-config>
</network-security-config>
```

上記の<pin>タグに記述するのは、ピンニング検証の対象となる公開鍵のハッシュ値を base64 でエンコードしたものである。また、ハッシュ関数は SHA-256 のみサポートされている。

非暗号化（**HTTP**）通信の抑制

Network Security Configuration を用いて、アプリの HTTP 通信（非暗号化通信）を抑制することができる。

非暗号化通信を抑制する方法は次の通りである：

1. 基本的には、*<base-config>* タグにより、すべてのドメインとの通信で非暗号化通信 (HTTP 通信) を抑制する[*14]
2. やむを得えず非暗号化通信が必要なドメインのみ *<domain-config>* タグにより個別に非暗号化通信を許可する例外設定をおこなう。なお、非暗号化通信を許可して良いかどうかの判断には「5.4.1.1. *HTTP* 通信する」を参考にすること

非暗号化通信の抑制は cleartextTrafficPermitted 属性を false に設定することで行われる。下にこの例を示す。

```
<?xml version="1.0" encoding="utf-8"?>
<network-security-config>
    <base-config cleartextTrafficPermitted="false">
    </base-config>
    <!-- 例外的に非暗号化通信を許可する必要のあるサイトを、<domain-config> タグにより明示的に "true"
に設定する -->
    <domain-config cleartextTrafficPermitted="true">
        <domain includeSubdomains="true">www.jssec.org</domain>
    </domain-config>
</network-security-config>
```

この設定は Android 8.0 (API Level 26) 以降では WebView にも適用されるが、Android 7.1 (API Level 25) 以前では WebView には適用されないことに注意する必要がある。

Android 9.0 (API Level 28) 未満では、属性 cleartextTrafficPermitted のデフォルト値が true であったが、Android 9.0 以降では false となった。従って API Level 28 以降をターゲットとしている場合は上の例の <base-config>での宣言は不要である。しかし意図を明確にし、またターゲット API Level による挙動の違いの影響を避けるため、上の例のように明示的に記載することを推奨する。

デバッグ専用のプライベート証明書

アプリ開発時にデバッグ目的でプライベート証明書を用いた開発用サーバーとの HTTPS 通信を行う場合、開発者は「5.4.3.3. 証明書検証を無効化する危険なコード」で述べたような、証明書検証を無効化させる危険な実装をアプリに組み込んでしまわないよう注意する必要がある。Network Security Configuration で以下のような設定を行えば、デバッグ時にのみ（AndroidManifest.xml 内の android:debuggable が”true” である場合のみ）使用する証明書を指定することがで

[*14] HTTP 以外の通信方式に対してどのような制御が行われるかについては、https://developer.android.com/reference/android/security/NetworkSecurityPolicy.html#isCleartextTrafficPermitted を参照すること。

きるため、前述のような危険なコードを製品版に残してしまう危険性がなくなり、脆弱性の作り込み防止に役立てることができる。

デバッグ時にのみプライベート証明書を用いる

```
<?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 （コラム） セキュア接続の TLS1.2/TLS1.3 への移行について

最初の公開版である 1994 年リリースの SSL2 にはセキュリティプロトコルとして重大な脆弱性があったため、根本から設計し直した SSL3.0(RFC6101) が 1995 年後半にリリースされた。しかし、2014 年に Google Security Team が発表した POODLE と呼ばれる脆弱性[*15] によって、SSL3.0 のパディングは安全ではないことが判明した。1999 年にリリースされた TLS1.0(RFC2246) ではパディングの設計上の欠陥は修正されているが、暗号化されたデータの一部を抜き出す攻撃手法 BEAST が、Duong と Rizzo により 2011 年に発表された[*16] 。セキュリティに関する修正 (TLS1.0 の安全性強化) を行なった TLS1.1(RFC4346) が 2006 年にリリースされ、更に 2008 年にリリースされた TLS1.2(RFC5246) ではハッシュ関数 SHA-2(SHA-256 や SHA-384) の利用、認証付暗号 (AEAD) 利用モード (GCM, CCM) が利用可能な暗号スイートのサポートなど、より強力な暗号アルゴリズムの利用が可能になっている。

このような背景から米国立標準技術研究所 (NIST) は、2018 年 10 月 15 日発行の TLS に関するガイドライン[*17] において TLS1.1 以前を非推奨あるいは禁止し、政府系だけではなく政府系以外のアプリケーションをサポートするサーバも TLS1.2 への移行を求めている。それにともない、 ここ数年のセキュリティ騒動や新しいバージョンの TLS の普及から「SSL や TLS の古いバージョン」のサポートを取りやめるサイトやサービスも増えており、TLS1.2 への移行が確実に進みつつある[*18] 。

移行を要件としている例として、クレジットカードセキュリティの国際標準化団体 PCI SSC によって策定された「ペイメントカード業界データセキュリティ基準 (PCI DSS: Payment Card Industry Data Security Standard)」と呼ばれるセキュリティ基準（最新版は 2018 年 5 月公開の v3.2.1）が挙げられる[*19] 。現在 E コマースはスマートフォンやタブレットでも広く利用されており、決済にはクレジットカードを利用する事が普通であろう。本書（Android アプリのセキュア設計・セキュアコーディングガイド）を利用するユーザーにおいても、アプリケーションを介してクレジットカード情報などをサーバサイドに送信するサービスを企画するケースも多いと予想するが、ネットワークでクレジットカードを利用する際にはその経路の安全性を確保することが必要で、PCI DSS は、このようなサービスにおける会員データを取り扱う際の基準であり、カードの不正利用や情報漏えいなどを防止する目的で定められている。このセキュリティ基準の中 では具体的なバージョンは明記していないが、SSL の全てのバージョンおよび既知のエクスプロイト（攻撃プログラム）に対して脆弱な TLS の初期バージョンを 2018 年 6 月 30 日までで廃止とし、安全とされるバージョン（すなわち

[*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] SSL/TLS 暗号設定ガイドライン, 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)

TLS1.2 以降）にアップグレードすることを要求している 。

スマートフォンとサーバー間の通信において、経路上の安全を確保することは、クレジット情報の取り扱いに限らずプライバシー情報のやりとりやその他情報のやりとりにおいても非常に重要な点であると言えるため、サービスを提供する (サーバー) 側が TLS1.2 を利用したセキュア接続へ移行する事は差し迫った課題であるといえる。

一方、クライアント側である Android において TLS 1.1 以降に対応できる WebView 機能は Android 4.4(Kitkat) 以降であり、HTTP 通信を直接行う場合は多少追加実装を伴うが Android 4.1(Jelly Bean 前期) からとなる。

サービス開発者の中には、TLS 1.2 を採用すると Android 4.3 以前のユーザーを切り捨てることになり、少なからず影響があると思われる向きがあるかもしれない。しかしながら、下図の通り、最新のデータ（2019 年 5 月現在）[*20] によると現在利用されている Android OS のシェアは 4.4 以降が 96.2% と圧倒的であるため、扱う資産などの安全性を考慮して、TLS 1.2 への移行を真剣に検討されることをお薦めする。

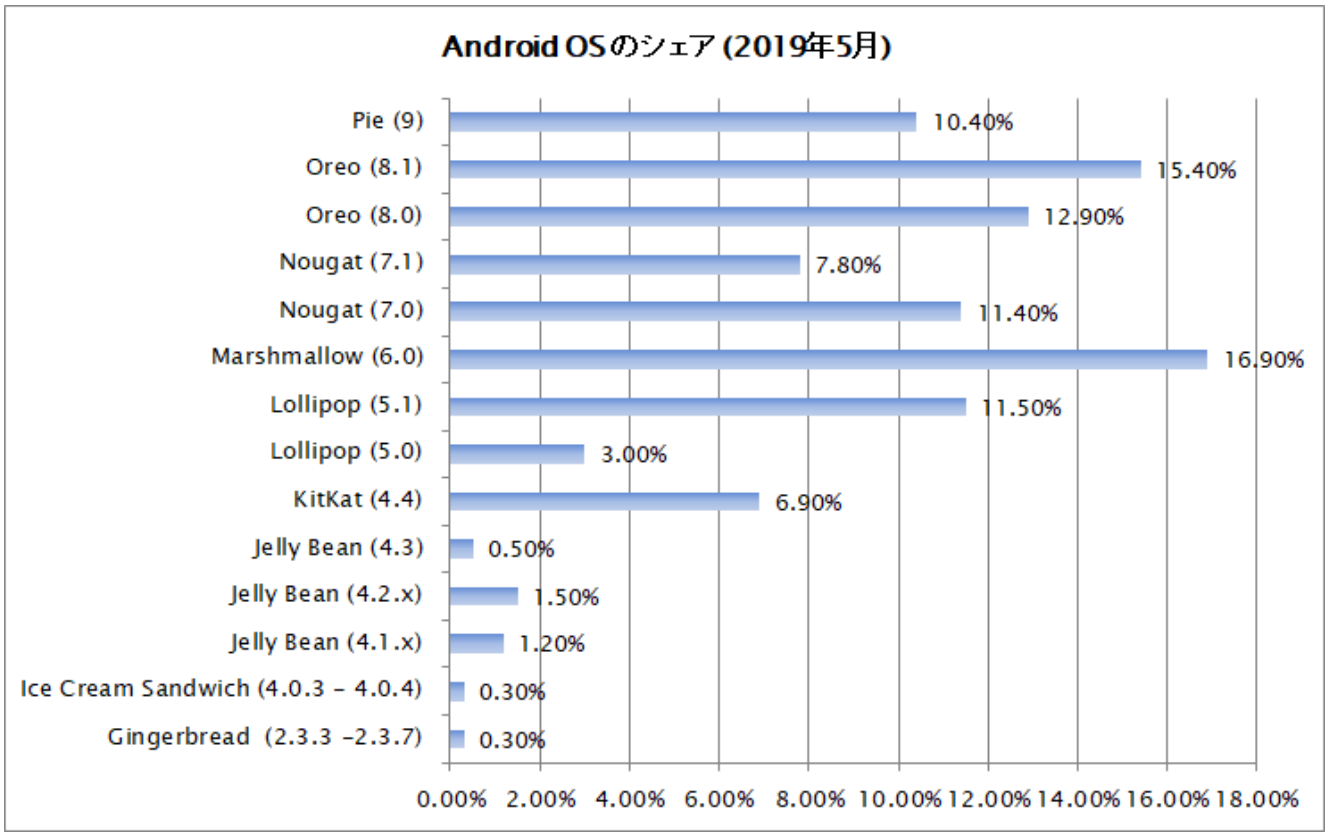

図 5.4.5　Android OS のシェア （Andorid Developpers サイトより）

2018 年 8 月に公開された TLS1.3(RFC8446) は、TLS1.2 策定以降に見つかった新たな脆弱性や攻撃手法への対策および性能向上を狙いとして、プロトコルと暗号アルゴリズムの抜本的な再設計が行われた[*21] 。Android 10 ではプラットフォームの TLS 実装が TLS 1.3 をサポートするようになり、すべての TLS 接続に対してデフォルトで TLS 1.3 が有効になった[*22] 。なお、Android 10 からは、安全性の低い下記の暗号スイートが削除された（モード：CBC、MAC:SHA2）[*23] 。

- TLS_RSA_WITH_AES_128_CBC_SHA256
- TLS_RSA_WITH_AES_256_CBC_SHA256

[*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_ECDSA_WITH_AES_128_CBC_SHA256
- TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA384
- TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256
- TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384

5.4.3.9 ECH（Encrypted Client Hello）の有効化

Android 17 以上をターゲットとするアプリでは、TLS ハンドシェイク時に Server Name Indication（SNI）を暗号化する TLS 拡張機能である ECH（Encrypted Client Hello）がデフォルトで有効になる。これまでの HTTPS 通信でも通信内容は暗号化されていたが、「どのサイトにアクセスしているか（ドメイン名）」を表す SNI は平文で送信されていたため、ドメイン名から利用しているサービス名などが推測される場合があった。Android 17 の ECH ではこのドメイン名部分も暗号化されるため、ネットワーク監視者がアプリの接続先の特定のドメインを簡単に特定することを防ぎ、ユーザーのプライバシーがさらに強化される。

アプリ内で TLS 接続時に ECH は自動利用されるが、実際に ECH が機能するには以下の条件を満たす必要がある。

- ネットワークライブラリが ECH 対応: HttpEngine、WebView、OkHttp などがサポートしていること。
- サーバーが ECH 対応: HTTPS DNS（HTTPS RR）に ECH 設定があり、利用している CDN（Cloudflare など）やリモートサーバーが ECH プロトコルをサポートしていること。
- TLS ネゴシエーション成功: 条件が揃えば自動的に ECH が使用される。揃わなければ通常の TLS にフォールバックされる（ネゴシエートできない場合、クライアントはランダム化されたコンテンツを含む ECH 拡張機能を送信する ECH GREASE と呼ばれるメカニズムが働く）。

表 5.4.3　ECH 有効化による通信の違い

項目	従来 TLS	Android 17 (ECH)
通信内容	暗号化	暗号化
ドメイン (SNI)	平文可視	暗号化
監視耐性	△	◎

デバッグやネットワーク監視への影響

セキュリティ製品などで URL フィルタリングしている場合やプロキシなどを利用している場合には、SNI が暗号化されることにより影響が出る可能性がある。また、開発環境の観点からも、通信ログ（pcap 等）からドメインの特定が不可能になるため、パケットキャプチャを用いたデバッグに影響が出るケースが考えられる。

ネットワークセキュリティ構成ファイルによるカスタマイズ

ECH は自動有効化され、アプリ開発者は基本的には設定不要であるが、デバッグ目的などでこの動作をカスタマイズしたい場合は、ネットワークセキュリティ構成ファイル（Network Security Configuration）を使用する。Android 17 では新たに <domainEncryption> 要素が追加されており、 <base-config> または <domain-config> タグ内でグローバルまたはドメインごとに ECH モード（ ”enabled” や ”disabled” など）を選択することができる。

```
<?xml version="1.0" encoding="utf-8"?>
<network-security-config>
    <domain-config>
```

(次のページに続く)

(前のページからの続き)

```
        <domain includeSubdomains="true">example.com</domain>
        <domainEncryption mode="enabled" />
    </domain-config>
</network-security-config>
```

ECH を確実に動作させるにはアプリ側だけでなくサーバーや DNS 側の対応が鍵となるため、ネットワーク全体の構成を確認することが望ましい。

5.4.3.10 証明書の透明性（CT）への対応

Android 17 以降をターゲットとするアプリでは、公開信頼された TLS 証明書に対する Certificate Transparency（CT）の検証がデフォルトで有効となる。TLS 証明書（HTTPS 用証明書）は、CA（認証局）が発行する仕組みになっているが、過去に誤発行・不正発行が発生した。その結果、MITM 攻撃等に晒されるリスクが問題視されており、その対策として CT が導入された。CT は、発行された証明書を公開ログ（CT ログ）に記録し、ログ登録証明として SCT（Signed Certificate Timestamp）を付与、クライアント（Android など）はこの SCT を検証する。つまり、証明書が公開ログに載っているかをチェックして不正な証明書を検出する仕組みである。

Android 16 以前では CT は自分で ON にしないと機能が使われないオプトイン形式であったが、Android 17 ではデフォルト有効化され暗黙的に適用されるため、開発者が意識しなくても検証が走る。CT 非対応証明書の場合、TLS 接続エラーや API 通信失敗、アプリ動作停止の可能性がある。

OS の挙動変更の検証例

実際に CT の検証が機能しているかは、SCT が含まれていないテスト用のサーバ（ https://no-sct.badssl.com など）へ接続を試みることで確認できる。Android 17 をターゲットとした環境（プロキシ等を介さないピュアなネットワーク環境）では通信がブロックされ、 SSLHandshakeException が発生して SCT の検証に失敗することが確認できる。

```
// CT の動作を検証するコード（抜粋）
private suspend fun testConnection(urlString: String, siteName: String): String {
    return withContext(Dispatchers.IO) {
        try {
            // "https://no-sct.badssl.com" などの SCT なしサイトへ接続を試みる
            val url = URL(urlString)
            val connection = url.openConnection() as HttpURLConnection
            connection.connectTimeout = 8000
            connection.readTimeout = 8000
            connection.connect()
            val responseCode = connection.responseCode
            connection.disconnect()
            "$siteName への接続に成功しました\n(HTTP Status: $responseCode)"
        } catch (e: SSLHandshakeException) {
            // Android 17（ターゲット SDK 37）では、SCT がないためここで SSLHandshakeException がスローされる
            "$siteName への接続に失敗しました\n 理由: SSLHandshakeException\n${e.message}"
        } catch (e: Exception) {
            "$siteName への接続に失敗しました\n 理由: ${e.javaClass.simpleName}: ${e.message}"
```

(次のページに続く)

(前のページからの続き)

```
        }
    }
}
```

デバッグやステージング環境への影響

実務的には本番環境の直前に用意される最終確認用のテスト環境であるステージング環境などで、self-signed 証明書（自己署名証明書）を利用する場合は注意が必要である。self-signed 証明書は信頼された CA ではなく自分自身で発行した証明書のため、CT ログに登録されず、Android 17 環境では通信エラーが発生する可能性がある。社内プロキシによる証明書差し替えが行われる開発環境でも同様である。

ネットワークセキュリティ構成ファイルによるカスタマイズ

開発者としては利用している証明書が CT 対応か確認することが重要となるが、DigiCert や Let's Encrypt 等の公的 CA は基本的には問題がないと考えられる。ステージング用環境を想定し自己署名証明書を利用する場合には、ネットワークセキュリティ構成ファイル（Network Security Configuration）で対象ドメインの CT を明示的にオフ (false) にするか、正規証明書を利用することが推奨される。

```
<domain-config>
    <domain includeSubdomains="true">staging.example.com</domain>
    <certificateTransparency enabled="false"/>
</domain-config>
```

CT 導入のメリットは、不正証明書の発見、MITM 攻撃の抑止、CA の透明性向上である。運用上の注意点としては、社内・検証環境が動作しなくなる可能性がある点や証明書管理の厳格化が必要な点である。Android 17 では証明書が CT ログに載っていることを前提にした世界になるが、全 HTTPS 通信に関係する機能であり広範囲に影響があるため開発者は注意が必要である。

5.4.3.11 usesCleartextTraffic の非推奨化と HTTP 通信の許可

Android 17 から、AndroidManifest.xml で HTTP 通信を許可するために用いられてきた usesCleartextTraffic 属性が非推奨（deprecated）となり、将来的な削除に向けて実質無効化への運用が開始された。

従来は usesCleartextTraffic="true" と設定することでアプリ全体の HTTP 通信が一律で許可されていた。しかし Android 17 以降では、このフラグがオンであっても、ネットワークセキュリティ構成ファイル（Network Security Configuration）が存在しない場合は HTTP 通信がブロックされる仕様に変更された。

今後 HTTP 通信を必要とする場合は、フラグ単体での許可に頼るのではなく、ネットワークセキュリティ構成ファイルを用いて、どのドメインとの通信を許可するのかを明示的に設定するセキュリティ設計が求められる。

ネットワークセキュリティ構成ファイルによる **HTTP** 通信の許可

特定のドメイン（例：example.com）に対する HTTP 通信を許可する場合、ネットワークセキュリティ構成ファイルを以下のように設定する。これにより、明示的に許可されたドメイン以外の HTTP 通信はブロックされる。

```
<network-security-config>
    <base-config cleartextTrafficPermitted="false"/>
    <domain-config cleartextTrafficPermitted="true">
        <domain>example.com</domain>
```

(次のページに続く)

(前のページからの続き)

```
    </domain-config>
</network-security-config>
```

OS の挙動変更の検証例

以下の検証コードは、ネットワークセキュリティ構成ファイルで HTTP 通信が許可されたドメイン（http://example.com）に対して GET リクエストを行う例である。ネットワークセキュリティ構成ファイルによる許可設定がない場合や、未許可のドメインに対して HTTP 通信を行った場合は、Android 17 の環境では通信がブロックされ、例外が発生する。

```
// HTTP 通信を行う検証コード（抜粋）
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"
        }
    } catch (e: Exception) {
        // ネットワークセキュリティ構成ファイルで許可されていない場合、
        // ここで例外（Cleartext communication to [ドメイン名] not permitted by network␣
↪security policy 等）がスローされる
        "Exception: ${e.message}"
    }
}
```

これまではフラグ 1 つで全ての HTTP 通信が許可される形であったが、Android 17 以降では「どの通信を許可するか」をドメイン単位で明示しないとブロックされるため、開発者はアプリが通信する外部接続先を正確に把握し、必要最小限の許可にとどめる厳格な管理が必要となる。

5.5 プライバシー情報を扱う

近年、プライバシー情報を守るための世界的な潮流として「プライバシー・バイ・デザイン」が提唱されており、この概念に基づき各国政府においてもプライバシー保護のための法制化を進めているところである。

スマートフォン内の利用者情報を活用するアプリは、利用者が個人情報やプライバシーの観点から安全・安心にアプリを活用できるように、利用者情報を適切に取り扱うとともに、利用者に対して分かりやすい説明を行い、利用者に利用の可否の選択を促すことが求められる。そのためには、アプリがどのような情報をどのように扱うか等を示したアプリ毎のアプリケーション・プライバシーポリシー（以下、アプリ・プライバシーポリシー）を作成・提示するとともに、慎重

な取り扱いが求められる利用者情報の取得・利用については事前にユーザーの同意を得る必要がある。なお、アプリケーション・プライバシーポリシーは従来から存在する「個人情報保護方針」や「利用規約」等とは異なり別途作成を要するものであることに留意すること。

プライバシーポリシーの作成や運用に関して、詳しくは総務省が提唱する『スマートフォンプライバシー イニシアティブ』、『スマートフォン プライバシーイニシアティブⅡ』及び『スマートフォン プライバシーイニシアティブⅢ』（以下総務省 SPI と省略）を参照のこと。

また、本記事で扱う用語については、本文内の解説および「5.5.3.2. 用語解説」を参照すること。

5.5.1 サンプルコード

アプリ・プライバシーポリシーの作成には、一般に公開されている「アプリケーション・プライバシーポリシー作成支援ツール[*24]」を利用することもできる。このツールの出力は HTML 形式および XML 形式となっており、概要版アプリケーション・プライバシーポリシーと詳細版アプリケーション・プライバシーポリシーのそれぞれのファイルが作成される。作成された XML ファイルには検査用のタグがつくなど、総務省 SPI に準拠した形となっている。以下のサンプルコードでは、上記ツールを使って作成した HTML ファイルを利用してアプリ・プライバシーポリシーを提示する例を示す。

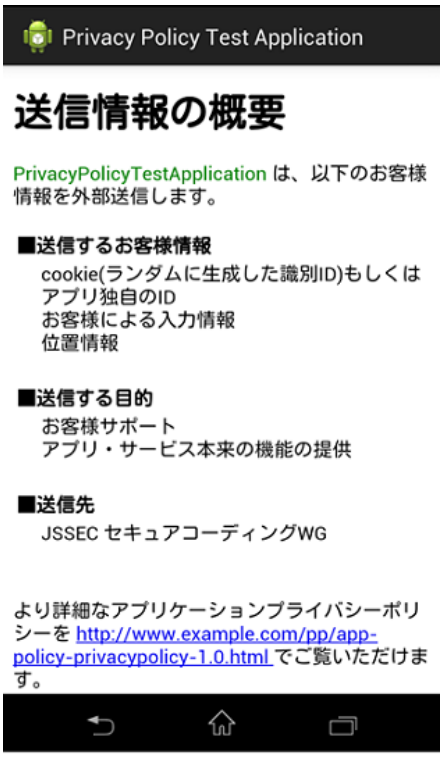

図 5.5.1　概要版アプリケーション・プライバシーポリシーの例

具体的には、次の判定フローに従うことで利用するサンプルコードを判断できる。

[*24] http://www.kddi-research.jp/newsrelease/2013/090401.html

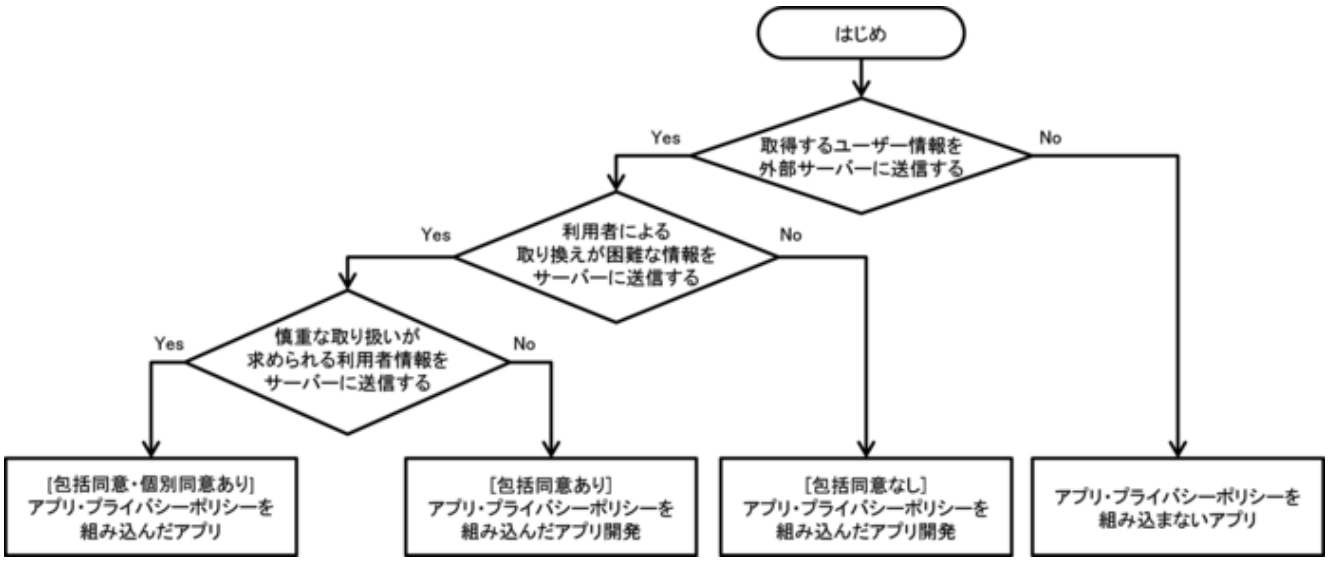

図 5.5.2　プライバシー情報を扱うサンプルコードを選択するフローチャート

ここで、包括同意とは、アプリ初回起動時のアプリ・プライバシーポリシーの提示・確認により、アプリがサーバーに送信する利用者情報について包括的に同意を得ることである。

また、個別同意とは、個々の利用者情報について送信の直前に個別の同意を得ることである。

5.5.1.1 [包括同意・個別同意あり] アプリ・プライバシーポリシーを組み込んだアプリ

ポイント：[包括同意・個別同意あり] アプリ・プライバシーポリシーを組み込んだアプリ

1. 初回起動時 (アップデート時) に、アプリが扱う利用者情報の送信について包括同意を得る
2. ユーザーの包括同意が得られていない場合は、利用者情報の送信はしない
3. 慎重な取り扱いが求められる利用者情報を送信する場合は、個別にユーザーの同意を得る
4. ユーザーの個別同意が得られていない場合は、該当情報の送信はしない
5. ユーザーがアプリ・プライバシーポリシーを確認できる手段を用意する
6. 送信した情報をユーザー操作により削除する手段を用意する
7. ユーザー操作により利用者情報の送信を停止する手段を用意する
8. 利用者情報の紐づけには UUID/cookie を利用する
9. アプリ・プライバシーポリシー概要版を assets フォルダ内に配置しておく

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
 *
```

(次のページに続く)

(前のページからの続き)

```
 *      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;
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 {
```

(次のページに続く)

(前のページからの続き)

```
    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) {
            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);
```

(次のページに続く)

(前のページからの続き)

```
        setContentView(R.layout.activity_main);

        mFusedLocationClient =
            LocationServices.getFusedLocationProviderClient(this);

        if (Build.VERSION.SDK_INT >= 23) {
            // 位置情報取得のためには、API Level 23 以上の場合パーミッションの確認が必要
            int permissionCheck =
                ContextCompat.checkSelfPermission(this,
                        Manifest.permission.ACCESS_FINE_LOCATION);
            if (permissionCheck != PackageManager.PERMISSION_GRANTED) {
                // パーミッションがないためユーザーの許可を得る
                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) {
                    // パーミッションが許可された
                    // ユーザー識別用 ID をサーバーから取得する
                    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 サービスが利用できない。サンプルアプリでは終了する。
                        finish();
                    }
                } else {
                    // パーミッションが許可されない。サンプルアプリでは終了する。
                    finish();
                }
            }
        }
```

(次のページに続く)

(前のページからの続き)

```
    }

    @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) {
            // ★ポイント 1★ 初回起動時（アップデート時）に、アプリが扱う利用者情報の送信について包括同
意を得る
            // アップデート時については新しい利用者情報を扱うようになった場合にのみ再度包括同意を得る必要
がある。
            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) {
        // 慎重な取り扱いが求められる利用者情報を送信について既に同意を得ているか確認する
        // 実際には送信する情報の種別毎に同意を得る必要があることに注意すること
        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) {
            // ★ポイント 3★ 慎重な取り扱いが求められる利用者情報を送信する場合は、個別にユーザーの同意
を得る
            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 {
            // 同意済みのため、送信処理を開始する
            onPositiveButtonClick(DIALOG_TYPE_PRE_CONFIRMATION);
```

(次のページに続く)

(前のページからの続き)

```
        }
    }

    public void onPositiveButtonClick(int type) {
        if (type == DIALOG_TYPE_COMPREHENSIVE_AGREEMENT) {
            // ★ポイント 1 ★ 初回起動時（アップデート時）に、アプリが扱う利用者情報の送信について包括同
意を得る
            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) {
            // ★ポイント 3 ★ 慎重な取り扱いが求められる利用者情報を送信する場合は、個別にユーザーの同意
を得る
            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();
                    }
                }
            });
            // 同意を得た旨、状態を保存する
            // 実際には送信する情報の種別毎に同意を得る必要があることに注意すること
            SharedPreferences.Editor pref =
                getSharedPreferences(PRIVACY_POLICY_PREF_NAME, MODE_PRIVATE).edit();
            pref.putInt(PRIVACY_POLICY_DISCRETE_TYPE1_AGREED_KEY, getVersionCode());
```

(次のページに続く)

(前のページからの続き)

```
            pref.apply();
        }
    }

    public void onNegativeButtonClick(int type) {
        if (type == DIALOG_TYPE_COMPREHENSIVE_AGREEMENT) {
            // ★ポイント 2★ ユーザーの包括同意が得られていない場合は、利用者情報の送信はしない
            // サンプルアプリではアプリケーションを終了する
            finish();
        } else if (type == DIALOG_TYPE_PRE_CONFIRMATION) {
            // ★ポイント 4★ ユーザーの個別同意が得られていない場合は、該当情報の送信はしない
            // ユーザー同意が得られなかったので何もしない
        }
    }

    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) {
            // 例外処理は割愛
        }
        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) {
            // ★ポイント 5★ ユーザーがアプリ・プライバシーポリシーを確認できる手段を用意する
            Intent intent = new Intent();
            intent.setClass(this, WebViewAssetsActivity.class);
            startActivity(intent);
            return true;
        } else if (itemId == R.id.action_del_id) {
```

(次のページに続く)

(前のページからの続き)

```
            // ★ポイント 6★ 送信した情報をユーザー操作により削除する手段を用意する
            new SendDataAsyncTack().execute(DEL_ID_URI, UserId);
            return true;
        } else if (itemId == R.id.action_donot_send_id) {
            // ★ポイント 7★ ユーザー操作により利用者情報の送信を停止する手段を用意する
            // 利用者情報の送信を停止した場合、包括同意に関する同意は破棄されたものとする
            SharedPreferences.Editor pref =
                getSharedPreferences(PRIVACY_POLICY_PREF_NAME, MODE_PRIVATE).edit();
            pref.putInt(PRIVACY_POLICY_COMPREHENSIVE_AGREED_KEY, 0);
            pref.apply();
            // 本サンプルでは利用者情報を送信しない場合、ユーザーに提供する機能が無くなるため
            // この段階でアプリを終了する。この処理はアプリ毎の都合に合わせて変更すること。
            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) {
            // ★ポイント 8★ 利用者情報の紐づけには UUID/cookie を利用する
            // 本サンプルではサーバー側で生成した ID を利用する
            SharedPreferences sp =
                getSharedPreferences(PRIVACY_POLICY_PREF_NAME, MODE_PRIVATE);
            UserId = sp.getString(ID_KEY, null);
            if (UserId == null) {
                // SharedPreferences 内にトークンが存在しなため、サーバーから ID を取り寄せる。
                try {
                    UserId = NetworkUtil.getCookie(GET_ID_URI, "", "id");
                } catch (IOException e) {
                    // 証明書エラーなどの例外をキャッチする
                    extMessage = e.toString();
                }
                // 取り寄せた ID を SharedPreferences に保存する。
                sp.edit().putString(ID_KEY, UserId).commit();
            }
            return UserId;
        }
```

(次のページに続く)

(前のページからの続き)

```
    @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;

        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) {
            // 証明書エラーなどの例外をキャッチする
            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();
```

(次のページに続く)

(前のページからの続き)

```
        }
    }
}
```

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;
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);
```

(次のページに続く)

(前のページからの続き)

```
    }

    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) {
        // ★ポイント1★ 初回起動時（アップデート時）に、アプリが扱う利用者情報の送信について包括同意を得る
        // ★ポイント3★ 慎重な取り扱いが求められる利用者情報を送信する場合は、個別にユーザーの同意を得る
        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) {
                // ★ポイント5★ ユーザーがアプリ・プライバシーポリシーを確認できる手段を用意する
                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.buttonOK,
                                  new DialogInterface.OnClickListener() {
            public void onClick(DialogInterface dialog, int whichButton) {
```

(次のページに続く)

(前のページからの続き)

```
                if (mListener != null) {
                    mListener.onPositiveButtonClick(type);
                }
            }
        });
        builder.setNegativeButton(R.string.buttonNG,
                                  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
```

(次のページに続く)

(前のページからの続き)

```
 *
 * 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.os.Bundle;
import android.webkit.WebSettings;
import android.webkit.WebView;

public class WebViewAssetsActivity extends Activity {
    // ★ポイント 9★ アプリ・プライバシーポリシー概要版を assets フォルダ内に配置しておく
    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 [包括同意あり] アプリ・プライバシーポリシーを組み込んだアプリ

ポイント：[包括同意あり] アプリ・プライバシーポリシーを組み込んだアプリ

1. 初回起動時 (アップデート時) に、アプリが扱う利用者情報の送信について包括同意を得る
2. ユーザーの包括同意が得られていない場合は、利用者情報の送信はしない
3. ユーザーがアプリ・プライバシーポリシーを確認できる手段を用意する
4. 送信した情報をユーザー操作により削除する手段を用意する
5. ユーザー操作により利用者情報の送信を停止する手段を用意する

6. 利用者情報の紐づけには UUID/cookie を利用する

7. アプリ・プライバシーポリシー概要版を assets フォルダ内に配置しておく

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.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;
```

(次のページに続く)

(前のページからの続き)

```
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 = "";

    private final int DIALOG_TYPE_COMPREHENSIVE_AGREEMENT = 1;

    private final int VERSION_TO_SHOW_COMPREHENSIVE_AGREEMENT_ANEW = 1;

    private final int MY_PERMISSIONS_READ_PHONE_NUMBERS = 257;

    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
```

(次のページに続く)

(前のページからの続き)

```
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        // ランダムに生成した UUID を送る
        mUUId = UUID.randomUUID().toString();

        // ユーザー識別用 ID をサーバーから取得する
        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) {
            // ★ポイント 1★ 初回起動時（アップデート時）に、アプリが扱う利用者情報の送信について包括同
意を得る
            // アップデート時については新しい利用者情報を扱うようになった場合にのみ再度包括同意を得る必要
がある。
            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();
```

(次のページに続く)

(前のページからの続き)

```
        new SendDataAsyncTack().execute(SEND_DATA_URI, UserId, nickname, mUUId);
    }

    public void onPositiveButtonClick(int type) {
        if (type == DIALOG_TYPE_COMPREHENSIVE_AGREEMENT) {
            // ★ポイント 1★ 初回起動時（アップデート時）に、アプリが扱う利用者情報の送信について包括同
意を得る
            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) {
            // ★ポイント 2★ ユーザーの包括同意が得られていない場合は、利用者情報の送信はしない
            // サンプルアプリではアプリケーションを終了する
            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) {
            // 例外処理は割愛
        }

        return versionCode;
    }

    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        getMenuInflater().inflate(R.menu.main, menu);
        return true;
    }

    @Override
    public boolean onOptionsItemSelected(MenuItem item) {
```

(次のページに続く)

(前のページからの続き)

```
        int itemId = item.getItemId();
            if (itemId == R.id.action_show_pp) {
                // ★ポイント 3★ ユーザーがアプリ・プライバシーポリシーを確認できる手段を用意する
                Intent intent = new Intent();
                intent.setClass(this, WebViewAssetsActivity.class);
                startActivity(intent);
                return true;
            } else if (itemId == R.id.action_del_id) {
                // ★ポイント 4★ 送信した情報をユーザー操作により削除する手段を用意する
                new SendDataAsyncTack().execute(DEL_ID_URI, UserId);
                return true;
            } else if (itemId == R.id.action_donot_send_id) {
                // ★ポイント 5★ ユーザー操作により利用者情報の送信を停止する手段を用意する

                // 利用者情報の送信を停止した場合、包括同意に関する同意は破棄されたものとする
                SharedPreferences.Editor pref =
                    getSharedPreferences(PRIVACY_POLICY_PREF_NAME, MODE_PRIVATE).edit();
                pref.putInt(PRIVACY_POLICY_AGREED_KEY, 0);
                pref.apply();

                // 本サンプルでは利用者情報を送信しない場合、ユーザーに提供する機能が無くなるため
                // この段階でアプリを終了する。この処理はアプリ毎の都合に合わせて変更すること。
                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) {
            // ★ポイント 6★ 利用者情報の紐づけには UUID/cookie を利用する
            // 本サンプルではサーバー側で生成した ID を利用する
            SharedPreferences sp =
                getSharedPreferences(PRIVACY_POLICY_PREF_NAME, MODE_PRIVATE);
            UserId = sp.getString(ID_KEY, null);
            if (UserId == null) {
                // SharedPreferences 内にトークンが存在しなため、サーバーから ID を取り寄せる。
                try {
```

(次のページに続く)

(前のページからの続き)

```
                UserId = NetworkUtil.getCookie(GET_ID_URI, "", "id");
            } catch (IOException e) {
                // 証明書エラーなどの例外をキャッチする
                extMessage = e.toString();
            }

            // 取り寄せた ID を SharedPreferences に保存する。
            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;
```

(次のページに続く)

(前のページからの続き)

```
            } catch (IOException e) {
                // 証明書エラーなどの例外をキャッチする
                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.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;
```

(次のページに続く)

(前のページからの続き)

```
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) {
        // ★ポイント 1★ 初回起動時（アップデート時）に、アプリが扱う利用者情報の送信について包括同意を得
る
        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) {
                // ★ポイント 3★ ユーザーがアプリ・プライバシーポリシーを確認できる手段を用意する
                Intent intent = new Intent();
                intent.setClass(getActivity(), WebViewAssetsActivity.class);
```

(次のページに続く)

(前のページからの続き)

```
                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.buttonOK,
                new DialogInterface.OnClickListener() {
                public void onClick(DialogInterface dialog, int whichButton) {
                    if (mListener != null) {
                        mListener.onPositiveButtonClick(type);
                    }
                }
            });
        builder.setNegativeButton(R.string.buttonNG,
                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;
```

(次のページに続く)

(前のページからの続き)

```
    }
}
```

WebViewAssetsActivity.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.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 {
    // ★ポイント 7★ アプリ・プライバシーポリシー概要版を assets フォルダ内に配置しておく
    private 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);
```

(次のページに続く)

(前のページからの続き)

```
        webView.loadUrl(ABST_PP_URL);
    }
}
```

5.5.1.3 [包括同意なし] アプリ・プライバシーポリシーを組み込んだアプリ

ポイント：[包括同意なし] アプリ・プライバシーポリシーを組み込んだアプリ

1. ユーザーがアプリ・プライバシーポリシーを確認できる手段を用意する
2. 送信した情報をユーザー操作により削除する手段を用意する
3. ユーザー操作により利用者情報の送信を停止する手段を用意する
4. 利用者情報の紐づけには UUID/cookie を利用する
5. アプリ・プライバシーポリシー概要版を assets フォルダ内に配置しておく

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;
```

(次のページに続く)

(前のページからの続き)

```
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 {
    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);

        // ユーザー識別用 ID をサーバーから取得する
        new GetDataAsyncTask().execute();
```

(次のページに続く)

(前のページからの続き)

```
        findViewById(R.id.buttonStart).setEnabled(false);
        ((TextView) findViewById(R.id.editTextNickname))
                                .addTextChangedListener(watchHandler);
    }

    public void onSendToServer(View view) {
        String nickname =
            ((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) {
            // ★ポイント 1★ ユーザーがアプリ・プライバシーポリシーを確認できる手段を用意する
            Intent intent = new Intent();
            intent.setClass(this, WebViewAssetsActivity.class);
            startActivity(intent);
            return true;
        } else if (itemId == R.id.action_del_id) {
            // ★ポイント 2★ 送信した情報をユーザー操作により削除する手段を用意する
            new sendDataAsyncTack().execute(DEL_ID_URI, UserId);
            return true;
        } else if (itemId == R.id.action_donot_send_id) {
            // ★ポイント 3★ ユーザー操作により利用者情報の送信を停止する手段を用意する

            // 本サンプルでは利用者情報を送信しない場合、ユーザーに提供する機能が無くなるため
            // この段階でアプリを終了する。この処理はアプリ毎の都合に合わせて変更すること。
            String message  = getString(R.string.stopSendUserData);
            Toast.makeText(MainActivity.this,
                    this.getClass().getSimpleName() + " - " + message,
                    Toast.LENGTH_SHORT).show();
            finish();
```

(次のページに続く)

(前のページからの続き)

```
            return true;
        }
        return false;
    }

    private class GetDataAsyncTask extends AsyncTask<String, Void, String> {
        private String extMessage = "";

        @Override
        protected String doInBackground(String... params) {
            // ★ポイント 4 ★ 利用者情報の紐づけには UUID/cookie を利用する
            // 本サンプルではサーバー側で生成した ID を利用する
            SharedPreferences sp =
                    getSharedPreferences(PRIVACY_POLICY_PREF_NAME, MODE_PRIVATE);
            UserId = sp.getString(ID_KEY, null);
            if (UserId == null) {
                // SharedPreferences 内にトークンが存在しなため、サーバーから ID を取り寄せる。
                try {
                    UserId = NetworkUtil.getCookie(GET_ID_URI, "", "id");
                } catch (IOException e) {
                    // 証明書エラーなどの例外をキャッチする
                    extMessage = e.toString();
                }

                // 取り寄せた ID を SharedPreferences に保存する。
                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];
```

(次のページに続く)

(前のページからの続き)

```
            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) {
                // 証明書エラーなどの例外をキャッチする
                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();
        }
    }
}
```

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
```

(次のページに続く)

(前のページからの続き)

```
 * 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 {
    // ★ポイント5★ アプリ・プライバシーポリシー概要版を assets フォルダ内に配置しておく
    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 アプリ・プライバシーポリシーを組み込まないアプリ

ポイント：アプリ・プライバシーポリシーを組み込まないアプリ

1. 取得した情報を端末内部でのみ利用する場合、アプリ・プライバシーポリシーを表示しなくても良い
2. マーケットプレイス等のアプリ説明欄に、取得した情報を外部送信しない旨を記載する

MainActivity.java

```
/*
 * Copyright (C) 2012-2026 Japan Smartphone Security Association
```

(次のページに続く)

(前のページからの続き)

```
 *
 * 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);
```

(次のページに続く)

(前のページからの続き)

```
        setContentView(R.layout.activity_main);

        mFusedLocationClient = LocationServices.getFusedLocationProviderClient(this);
        if (Build.VERSION.SDK_INT >= 23) {
            // 位置情報取得のためには、API Level 23 以上の場合パーミッションの確認が必要
            Boolean permissionCheck =
                (ContextCompat.checkSelfPermission(this,
                    Manifest.permission.ACCESS_FINE_LOCATION)
                    == PackageManager.PERMISSION_GRANTED);
            if (!permissionCheck) {
                // パーミッションがないためユーザーの許可を得る
                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) {
                    // パーミッションが許可された
                    int resultCode =
                        GoogleApiAvailability.getInstance()
                                             .isGooglePlayServicesAvailable(this);
                    if (resultCode != ConnectionResult.SUCCESS) {
                        // Googleplay サービスが利用できない。サンプルアプリでは終了する。
                        finish();
                    }
                } else {
                    // パーミッションが許可されない。サンプルアプリでは終了する。
                    finish();
                }
            }
        }
    }

    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        getMenuInflater().inflate(R.menu.main, menu);
        return true;
```

(次のページに続く)

(前のページからの続き)

```
    }

    public void onStartMap(View view) {
        Log.d("onStartMap()"," called");
        // ★ポイント 1 ★ 取得した情報を端末内部でのみ利用する場合、アプリ・プライバシーポリシーを表示しなくても良い
        mFusedLocationClient.getLastLocation()
            .addOnSuccessListener(this, new OnSuccessListener<Location>() {
            @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);
                }
            }
        });
    }
}
```

★ポイント 2 ★マーケットプレイス等のアプリ説明欄に、取得した情報を外部送信しない旨を記載する

図 5.5.3　マーケットプレイス上での説明例

5.5.2 ルールブック

プライバシー情報を扱う際には以下のルールを守ること。

1. 送信する利用者情報は必要最低限に留める （必須）
2. 初回起動時 *(*アップデート時*)* に、ユーザーによる取り換えが困難、または、慎重な取り扱いが求められる利用者情報の送信について包括同意を得る （必須）
3. 慎重な取り扱いが求められる利用者情報を送信する場合は、ユーザーに個別同意を得る （必須）
4. ユーザーがアプリ・プライバシーポリシーを確認できる手段を用意する （必須）
5. 利用者情報と紐づける *ID* には *UUID/cookie* を用いる *(*端末固有 *ID* を用いない*)* （必須）
6. 送信した利用者情報をユーザー操作により削除および送信停止する手段を用意する （推奨）
7. 利用者情報を端末内のみで利用する場合、外部送信しない旨をユーザーに通知する （推奨）
8. アプリ・プライバシーポリシー概要版を *assets* フォルダ内に配置しておく （推奨）

5.5.2.1 送信する利用者情報は必要最低限に留める （必須）

アプリマニュアルなどの説明を元にユーザーが想起できるアプリの動作および目的以外で利用者情報にアクセスしないよう設計すること。また、アプリの動作および目的に必要な利用者情報であっても、アプリ内部でのみ必要な情報と外部サーバー等に送信すべき情報を精査し、必要最低限の情報のみを送信すること。

例えば、アラームアプリで、指定した時間にアラームが鳴る機能しか提供していないにもかかわらず位置情報をサーバーに送信している場合、ユーザーは位置情報とアプリの機能を関連付けることは難しい。一方で、このアラームアプリが、ユーザーがいる場所に応じて設定されたアラームを鳴らす機能を持つ場合は、アプリが位置情報を利用する理由を説明することができる。このように、ユーザーが機能と利用者情報との関連を理解できる説明が可能であれば、利用者情報の利用に妥当性があるとみなせる。

5.5.2.2 初回起動時 (アップデート時) に、ユーザーによる取り換えが困難、または、慎重な取り扱いが求められる利用者情報の送信について包括同意を得る （必須）

ユーザーによる取り換えが困難な利用者情報や慎重な取り扱いが求められる利用者情報を外部サーバーに送信する場合、ユーザーがアプリを使用する前に、アプリがどのような情報をどのような目的でサーバーに送信するか、第三者提供はあるかなどについて、事前にユーザーの同意 (オプトイン) を得ることが求められる。具体的には、アプリの初回起動時にアプリ・プライバシーポリシーを提示して、ユーザーの確認と同意を得るようにするべきである。また、アプリケーションのアップデートにより、新たな利用者情報を外部サーバーに送信するようになった時にも、再度ユーザーの確認と同意を得るようにする必要がある。同意が得られない場合は、アプリを終了するなど、情報の送信が必要な機能 (処理) を無効にすること。

これにより、ユーザーが利用者情報の利用状況を理解した上でアプリを使用していることが保証され、ユーザーに安心感を提供するとともにアプリへの信頼感を得ることが期待できる。

MainActivity.java

```
protected void onStart() {
    super.onStart();
```

(次のページに続く)

(前のページからの続き)

```
    // ～省略～
    if (privacyPolicyAgreed <= VERSION_TO_SHOW_COMPREHENSIVE_AGREEMENT_ANEW) {
        // ★ポイント★ 初回起動時（アップデート時）に、ユーザーによる取り換えが困難、または、
        // 慎重な取り扱いが求められる利用者情報の送信について包括同意を得る
        // アップデート時については、新しい利用者情報を扱うようになった場合にのみ、再度包括同意を得る必要
がある。
        ConfirmFragment dialog =
                ConfirmFragment.newInstance(R.string.privacyPolicy,
                                            R.string.agreePrivacyPolicy,
                                            DIALOG_TYPE_COMPREHENSIVE_AGREEMENT);
        dialog.setDialogListener(this);
        FragmentManager fragmentManager = getSupportFragmentManager();
        dialog.show(fragmentManager, "dialog");
    }
```

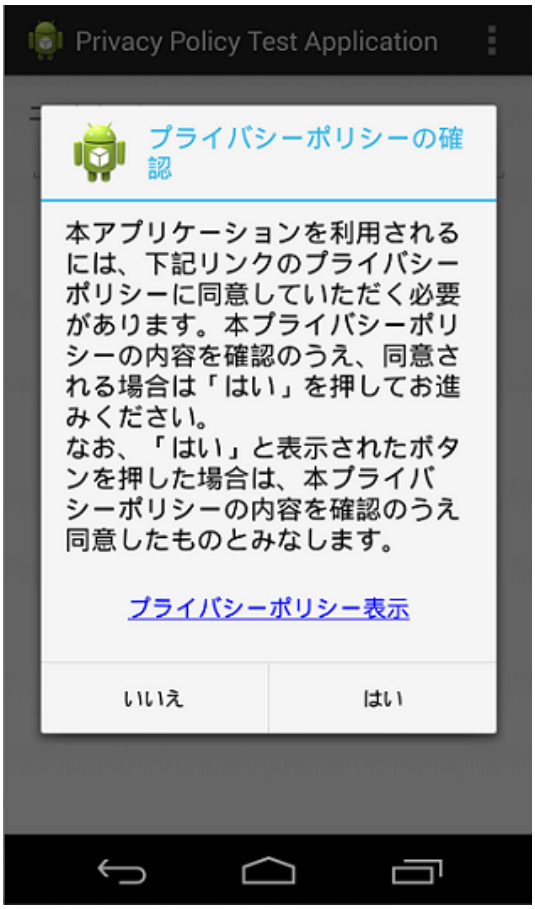

図 5.5.4　包括同意の例

5.5.2.3 慎重な取り扱いが求められる利用者情報を送信する場合は、ユーザーに個別同意を得る （必須）

慎重な取扱いが求められる利用者情報を外部サーバーに送信する場合は、包括同意に加えて、利用者情報 (または利用者情報の送信を伴う機能) 毎に、事前にユーザーの同意 (オプトイン) を得ることが必要である。ユーザーの同意が得られなかった場合は、外部サーバーへの情報送信は実施してはならない。

これにより、ユーザーは包括同意で確認した利用者情報の送信に関して、より具体的なアプリ機能 (提供サービス) との

関連を知ることができ、アプリ提供者はユーザーのより正確な判断による同意を得ることが期待できる。

```
MainActivity.java
public void onSendToServer(View view) {
    // ★ポイント★慎重な取り扱いが求められる利用者情報を送信する場合は、ユーザーに個別同意を得る
    ConfirmFragment dialog =
        ConfirmFragment.newInstance(R.string.sendLocation,
                                    R.string.cofirmSendLocation,
                                    DIALOG_TYPE_PRE_CONFIRMATION);
    dialog.setDialogListener(this);
    FragmentManager fragmentManager = getSupportFragmentManager();
    dialog.show(fragmentManager, "dialog");
}
```

図 5.5.5　個別同意の例

5.5.2.4 ユーザーがアプリ・プライバシーポリシーを確認できる手段を用意する （必須）

一般に、ユーザーが該当アプリをインストールする前にアプリ・プライバシーポリシーを確認する手段として、Android アプリのマーケットプレイス上にアプリ・プライバシーポリシーのリンクを表示する機能がある。この機能への対応に加え、アプリを端末にインストールした後でも、ユーザーがアプリ・プライバシーポリシーを参照できる手段を用意すること。特に利用者情報を外部サーバー等に送信する場合の個別同意確認時にはアプリ・プライバシーポリシーを容易に確認できる手段を用意し、ユーザーが正しく判断できるようにすることが望ましい。

```
MainActivity.java
    @Override
    public boolean onOptionsItemSelected(MenuItem item) {
        switch (item.getItemId()) {
        case R.id.action_show_pp:
            // ★ポイント★ ユーザーがアプリ・プライバシーポリシーを確認できる手段を用意する
            Intent intent = new Intent();
            intent.setClass(this, WebViewAssetsActivity.class);
            startActivity(intent);
            return true;
```

図 5.5.6　プライバシーポリシー表示用メニュー

5.5.2.5 利用者情報と紐づける ID には UUID/cookie を用いる (端末固有 ID を用いない)（必須）

IMEI などの端末固有 ID は、利用者情報と紐づけて送信すべきでない。端末固有 ID と利用者情報が紐づいた形で一度でも公開や漏洩してしまうと、後から端末固有 ID の変更が不可能なため、ID と利用者情報の紐づけを切ることができない（難しい）ことが、その理由である。また、Android 10 では、アプリの targetSdkVersion に関わらず、IMEI などの端末固有 ID を取得できなくなっている。そのため、端末固有 ID を用いず、UUID/cookie などの乱数をベースにした都度作成する変更可能な ID を使って、利用者情報との紐づけおよび送信をすること。これにより、下記で説明する『忘れら

れる権利』を考慮した実装とすることもできる。

```
MainActivity.java
    @Override
    protected String doInBackground(String... params) {
        // ★ポイント★ 利用者情報の紐づけには UUID/cookie を利用する
        // 本サンプルではサーバー側で生成した ID を利用する
        SharedPreferences sp =
            getSharedPreferences(PRIVACY_POLICY_PREF_NAME, MODE_PRIVATE);
        UserId = sp.getString(ID_KEY, null);
        if (UserId == null) {
            // SharedPreferences 内にトークンが存在しなため、サーバーから ID を取り寄せる。
            try {
                UserId = NetworkUtil.getCookie(GET_ID_URI, "", "id");
            } catch (IOException e) {
                // 証明書エラーなどの例外をキャッチする
                extMessage = e.toString();
            }

            // 取り寄せた ID を SharedPreferences に保存する。
            sp.edit().putString(ID_KEY, UserId).commit();

        return UserId;
    }
```

5.5.2.6 アプリ・プライバシーポリシー概要版を **assets** フォルダ内に配置しておく （推奨）

アプリ・プライバシーポリシーの概要版は assets フォルダ内に配置しておき、必要に応じて閲覧に利用することが望ましい。アプリ・プライバシーポリシーが assets フォルダにあれば、いつでも容易にアクセス可能であると同時に、悪意のある第三者により、偽造や改造されたアプリ・プライバシーポリシーを参照させられるリスクが回避可能なためである。また、assets フォルダに配置するファイルは、利用者情報に関する第三者検証が可能であることが望ましい。本節で紹介した「アプリケーション・プライバシーポリシー作成支援ツール」を利用する場合、検査用タグのある XML ファイルも assets フォルダに配置することで第三者検証が可能になる。

5.5.2.7 送信した利用者情報をユーザー操作により削除および送信停止する手段を用意する （推奨）

利用者の求めに応じ、外部サーバー上に送信された利用者情報を削除する機能を提供することが望ましい。同様に、アプリ自身で端末内に利用者情報（もしくはそのコピー）を保存した場合も、その情報を削除する機能をユーザーに提供することが望ましい。また、利用者の求めに応じ、利用者情報の送信を停止する機能を提供することが望ましい。

本ルール（推奨）の内、利用者情報の削除については、EU で提唱されている『忘れられる権利』により定められたものであるが、今後は利用者データ保護に関する個人の権利の強化も提案されているため、特に理由がない限り、本ガイドでは利用者情報の削除機能を用意することを推奨する。また、利用者情報の送信停止については、主にブラウザによる対応が進んでいる『Do Not Track（追跡拒否）』の観点により定められたものであり、削除同様に送信停止機能を用意することを推奨する。

```
MainActivity.java
    @Override
    public boolean onOptionsItemSelected(MenuItem item) {
        switch (item.getItemId()) {
        // ～省略～
        case R.id.action_del_id:
            // ★ポイント★ 送信した利用者情報をユーザー操作により削除および送信停止する手段を用意する
            new SendDataAsyncTack().execute(DEL_ID_URI, UserId);
            return true;
        }
```

5.5.2.8 利用者情報を端末内のみで利用する場合、外部送信しない旨をユーザーに通知する （推奨）

利用者の端末内部で一時的に利用者情報にアクセスするのみの場合であっても、利用者の理解を助け透明性を高めるために、その旨を伝えることが望ましい。具体的には、アクセスした利用者情報は、ある決まった目的のために端末内部で一時的に使用するのみであり、端末内に保存したり、外部サーバに送信したりしない旨をを利用者に通知すると良い。通知方法としては、マーケットプレイス上でのアプリ説明欄に記載するなどの方法が考えられる。なお、端末内での一時利用のみの場合は、アプリ・プライバシーポリシーへの記載は必須ではない。

図 5.5.7　マーケットプレイス上での説明例

5.5.3 アドバンスト

5.5.3.1 プライバシーポリシーを取りまく状況

スマートフォンにおける利用者情報を取得し、外部に送信する場合には、利用者に対してアプリがどのような情報をどのように扱うか等を示したアプリ・プライバシーポリシーを作成・提示する必要がある。アプリ・プライバシーポリシーに記載すべき内容は、総務省 SPI にて定義されている。

アプリ・プライバシーポリシーは個々のアプリが扱うすべての利用者情報と、その用途や情報の保存先、送信先等を明らかにすることに主眼が置かれている。それとは別に事業者が個々のアプリから収集したすべての利用者情報をどのように保管・管理・廃棄をするかを示す事業者プライバシーポリシーも必要となる。従来、個人情報保護法をもとに作成されているプライバシーポリシーとはこの事業者プライバシーポリシーが相当する。

プライバシーポリシーの作成・提示方法やそれぞれのプラバシーポリシーの役割分担などは、『JSSEC スマホ・アプリのプライバシーポリシー作成・開示についての考察』(https://www.jssec.org/event/20140206/03-1_app_policy.pdf) に詳細説明が記載されているので参照のこと。

5.5.3.2 用語解説

以下に、本ガイドで使用している用語について、『JSSEC スマホ・アプリのプライバシーポリシー作成・開示についての考察』(https://www.jssec.org/event/20140206/03-1_app_policy.pdf) に記載されている用語解説を引用しておく。

表 5.5.1　用語解説

用語	解説
事業者プライバシーポリシー	事業者における個人情報保護方針を記したプライバシーポリシー。個人情報保護法に基づき作成。
アプリ・プライバシーポリシー	アプリケーション向けプライバシーポリシー。総務省 SPI の指針に基づき作成。概要版と詳細版での解りやすい説明が望まれる。
概要版アプリ・プライバシーポリシー	アプリが取得する利用者情報、取得目的、第三者提供の有無などを簡潔に記述した文書。
詳細版アプリ・プライバシーポリシー	総務省 SPI の定める８項目に準拠した詳細な内容を記述した文書。
利用者による取り換えが容易な利用者情報	cookie、UUID など。
利用者による取り換えが困難な利用者情報	IMEI、IMSI、ICCID、MAC アドレス、OS が生成する ID など。
慎重な取り扱いが求められる利用者情報	位置情報、アドレス帳、電話番号、メールアドレスなど。

5.5.3.3 Android ID のバージョンによる違いについて

Android ID(Settings.Secure.ANDROID_ID) は、ランダムに生成される 64 ビットの数値を 16 進数文字列で表現したものであり、(ごく稀に重複する可能性はあるが) 端末を個別に識別することができる識別子である。そのため、使用法を間違えるとユーザーのトラッキングに繋がるリスクが大きくなり、使用の際には注意を必要とするが、Android 7.1(API Level 25) 以前と Android 8.0(API Level26) 以降の端末では、生成規則やアクセス可能範囲などが異なるため以下ではその違いについて説明する。

Android 7.1(API Level 25) 以前の端末

Android 7.1(API Level 25) 以前の端末では端末内に 1 つの値を持ちすべてのアプリからアクセス可能となる。ただし、マルチユーザーをサポートする端末ではユーザー毎に別の値が生成される。生成のタイミングとしては、最初は端末の工場出荷後の初回起動時であり、その後は、ファクトリーリセットの度に新しく生成される。

Android 8.0(API Level 26) 以降の端末

Android 8.0(API Level 26) 以降の端末ではアプリ (開発者) 毎に異なる値を持ち、対象のアプリだけがアクセスできるように変更されている。具体的には、Android 7.1(API Level 25) 以前のユーザーおよび端末に加えて、アプリの署名を要素として値が一意に決定される仕組みとなっており、署名の異なるアプリは別の (署名が同じ場合は同じ)Android ID の値を持つことになる。

値が変更されるタイミングに関しては以前とほぼ変わらないが、いくつか注意する点があるので以下に示す。

- パッケージのアンインストール・再インストール時

Android ID の値は、署名が同じである限り、アプリをアンインストール・再インストールされても変わらない。逆に、署名する鍵が変わった場合は、パッケージ名が同じでも再インストール時に Android ID の値が異なることに注意する。

- Android 8.0(API Level 26) 以降の端末へのアップデート時

Android 7.1(API Level 25) 以前の端末で既にアプリがインストールされていた場合、端末を Android 8.0(API Level 26) 以降にアップデートしてもアプリで取得できる Android ID の値は変わらない。ただし、アップデート後にアプリがアンインストール・再インストールする場合は除く。

なお、いずれの Android ID でも「5.5.3.2. 用語解説」における「利用者による取り換えが困難な利用者情報」に分類されるため、冒頭に言及したように使用の際は同様の注意を払うことを推奨する。

5.5.3.4 Android 10 での再設定不可能なデバイス ID の取得制限について

Android 10 では、プライバシー保護のため、再設定不可能なデバイス ID の取得がより制限されることになった。取得するためには、READ_PRIVILEGED_PHONE_STATE パーミッションが必要であるが、このパーミッションは通常のアプリには付与されない。この変更は、targetSdkVersion の指定に限らず、Android 10 上で動く全てのアプリに影響する。そのため、通常パーミッションが付与され動作していたアプリでも、SecurityException の発生や null の返却により想定している挙動とならない可能性がある。以下に、そのような影響を受ける情報の種類と取得するための API を示す。(READ_PHONE_STATE のような必要なパーミッションは付与されているものとする)

- Build クラス

表 5.5.2　Build クラス

API	取得情報	targetSDKVersion=29	targetSDKVersion ＜ 29
(フィールド)BUILD.SERIAL	デバイスのシリアル番号	Unknown	Unknown
getSerial	デバイスのシリアル番号	SecurityException	Unknown

- TelephonyManager クラス

表 5.5.3　TelephonyManager クラス

API	取得情報	targetSDKVersion=29	targetSDKVersion ＜ 29
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 データアクセスの監査について

Android 11 では位置情報や連絡先リストなどユーザーのプライベートデータにアクセスするプロセスに透明性をもたせるため、「データアクセスの監査」が導入されている。利用するには AppOpsManager.OnOpNotedCallback のインスタンスを登録し、アクティビティの onCreate() メソッド内など、データアクセスを監査するコンポーネントにコールバックロジックを実装する。これにより、ユーザーのプライベートデータへのアクセス記録が容易となる。

アクセス記録には定義したアトリビューションタグを含めることもでき、定義したアトリビューションタグの名前がアプリのログに含まれるようコールバックロジックを定義することもできる。

```
public class SharePhotoLocationActivity extends AppCompatActivity {
    private Context attributionContext;
    // ～省略～
    @Override
    public void onCreate(@Nullable Bundle savedInstanceState,
                         @Nullable PersistableBundle persistentState) {
        attributionContext = createAttributionContext("sharePhotos");
        // ～省略～
        AppOpsManager.OnOpNotedCallback appOpsCallback =
            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());
```

(次のページに続く)

（前のページからの続き）

```
                }
            };

        AppOpsManager appOpsManager = getSystemService(AppOpsManager.class);
        if (appOpsManager != null) {
            appOpsManager.setNotedAppOpsCollector(appOpsCollector);
        }
    }
```

また、Android 12 をターゲットとするアプリの場合はこれらのアトリビューションタグをマニフェストファイル内で宣言する必要がある。

```
<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>
```

宣言していないアトリビューション タグを使用しようとすると、attributionTag の値は null となった状態で Logcat に出力される。

5.5.3.6 位置情報アクセスについて

Android OS における位置情報アクセスは、Android 10 でのフォアグラウンドとバックグラウンドの分離をはじめ、段階的な権限リクエスト（Android 11）、おおよその位置情報の選択（Android 12）、Wi-Fi デバイス検出の権限分離（Android 13）など、バージョンを重ねるごとにプライバシー保護を目的とした権限モデルの細分化と厳格化が進められてきた。

しかし、これまでの権限モデルでは、位置情報を求めるたびに「アプリの使用中のみ許可」「今回のみ」「許可しない」といった 3 択ダイアログが表示されており、この選択肢はユーザーに都度判断を強いるため認知的負荷を与え、「ダイアログ疲れ」を引き起こす原因ともなっていた。

Android 17 における「位置情報ボタン」の導入

Android 17 で新たに導入された「位置情報ボタン」は、ユーザー体験（UX）とプライバシー保護を両立させる画期的な機能である。画面上に配置された「位置情報ボタン」をユーザー自らがタップする方式に変わることで、意思表示と権限付与が 1 ステップで完結し、操作のスムーズさが劇的に向上している。

この機能がもたらす最大のメリットは、「常時許可」から「データ最小化」への転換である。ボタンを 1 回タップするだけで、その時限りの正確な位置情報アクセスが許可される。例えば「SNS の投稿に場所をタグ付けする」「近くの店舗を検索する」といった操作では、タスクが完了すると権限も自動的に消失するため、ユーザーに対して「アプリが裏で勝手に追跡していない」という強い安心感を与えることができる。

位置情報ボタンは、Google が提供する Android Jetpack のライブラリとして提供されるため、既存の画面レイアウトへ手軽に配置できる。また、高い後方互換性を持っており、Android 16 以前のデバイスでユーザーがこのボタンをタップ

した場合には、自動的に従来型の位置情報リクエスト用ダイアログが表示される仕組みになっている。開発者は OS のバージョンごとに複雑な条件分岐を実装する必要がなく、シンプルなコードで最新の UI と従来の挙動を両立することが可能である。

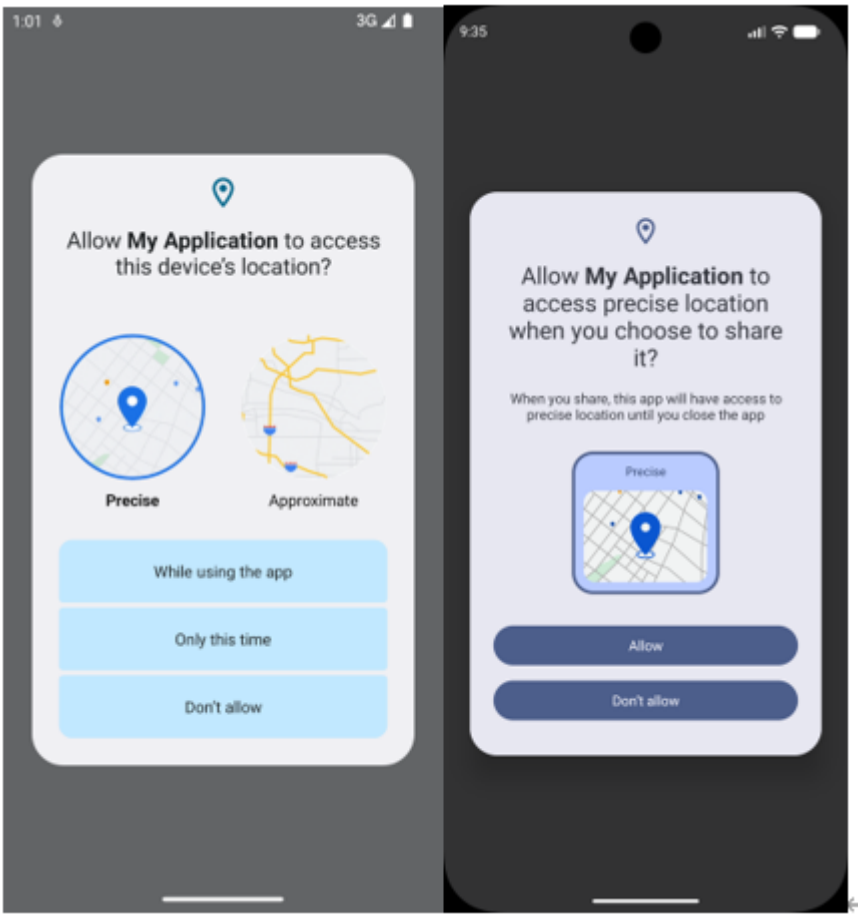

図 5.5.8　Android16 以下のデバイスで動作させた時の画面（左）、Android17 のデバイスで動作させた時の画面（右）

5.5.3.7 ローカル ネットワークへのアクセス権

従来は INTERNET 権限さえあればアプリが家庭内ネットワークの機器に自由にアクセスできたが、これにはユーザーの環境や利用状況の推測、個人識別、IoT 機器の乗っ取りといったリスクがあった。この仕様変更の背景には、そうした不正なアクセスからユーザーのプライバシーとセキュリティを守る目的がある。ローカルネットワーク制限によって、悪意のあるアプリが無制限の LAN アクセスを利用してユーザートラッキングやフィンガープリント採取を密かに行うことを防ぎ、ユーザーが明示的に許可しない限りアプリが LAN 内の通信を行えなくなる。

Android 16 におけるローカルネットワーク制限（オプトイン）

Android 16 では、アプリがローカルネットワーク（LAN）上の機器と通信する際に NEARBY_WIFI_DEVICES 権限が必要になる制限が導入された。もともと Android 13 で登場したこの権限の適用範囲が拡張され、LAN 上の IP アドレス（192.168.x.x や 10.x.x.x など）への TCP/UDP 通信や、mDNS や NSD などのローカルネットワークサービス探索も含めて、権限がなければアクセスできなくなる。ただし、この制限はデフォルトではオフになっており、開発者が adb コマンドで RESTRICT_LOCAL_NETWORK フラグを明示的に有効化した場合のみ適用される（オプトイン方式）。

Android 17 におけるローカルネットワーク権限の必須化

Android 17 以降では、この制限が全アプリに強制適用される。新たに ACCESS_LOCAL_NETWORK ランタイム権限が導入され、リクエストが必須となった。アプリが同じ Wi-Fi 内の機器（スマートホームデバイス、メディアキャスト、

ファイル共有など）と通信する場合、この権限を宣言して実行時にユーザーの許可を取得する必要がある。なお、この権限は既存の NEARBY_DEVICES 権限グループに該当するため、他の NEARBY_DEVICES 権限をすでに付与しているユーザーには、再度権限の付与を求めるプロンプトは表示されない。

今後、全アプリに強制適用される見込みなので、LAN アクセスを必要とするアプリは早めに対応・検証しておく必要がある。

実装と検証方法

まず、AndroidManifest.xml に Android 16 用の NEARBY_WIFI_DEVICES と Android 17 用の ACCESS_LOCAL_NETWORK 、および INTERNET などのパーミッションを宣言する。

```
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />

<!-- Android 16用（オプトイン検証用） -->
<uses-permission android:name="android.permission.NEARBY_WIFI_DEVICES"
    android:usesPermissionFlags="neverForLocation"
    tools:targetApi="s" />

<!-- Android 17以上 ランタイムパーミッション -->
<uses-permission android:name="android.permission.ACCESS_LOCAL_NETWORK" />
```

Android 16 環境でローカルネットワーク制限を検証する場合は、adb コマンドで制限を有効化し、端末を再起動する。

```
adb shell am compat enable RESTRICT_LOCAL_NETWORK <package_name>
adb reboot
```

アプリ側では、たとえば Jetpack Compose を使って以下のようなコードで接続を検証できる。ボタンを押すと LAN 内の IP アドレスにソケット接続を試み、その結果を画面に表示する。

```
Button(
    onClick = {
        testResult = "テスト中..."
        Thread {
            try {
                Socket().use { socket ->
                    socket.connect(InetSocketAddress("192.168.3.1", 80), 2000)
                }
                (context as? ComponentActivity)?.runOnUiThread {
                    testResult = "接続成功"
                }
            } catch (e: Exception) {
                (context as? ComponentActivity)?.runOnUiThread {
                    testResult = "接続失敗: ${e.localizedMessage}"
                }
            }
        }.start()
```

（次のページに続く）

(前のページからの続き)

```
    },
    modifier = Modifier.fillMaxWidth()
) {
    Text("ローカルネットワーク接続テスト")
}
```

権限が拒否されている状態（または Android 16 でオプトインし、権限を付与していない状態）でこのボタンを押すと、sendto failed: EPERM (Operation not permitted) や sendto failed: ECONNABORTED (Operation not permitted) といったエラーが返る。実行時に権限を適切にリクエストし許可を得ると、LAN アクセスが復活し、正常に通信できるようになる。

5.5.3.8 Android 12 における使用していないアプリの自動休止機能について

Android 12(API Level 31) をターゲットとするアプリの場合、Android 11 から導入された権限の自動リセットに加え、一定期間使用していないアプリについては自動休止機能が適用される。これには以下のような特徴がある。

- アプリのキャッシュ内のファイルはすべて削除されパフォーマンスではなく保存容量に合わせて最適化がおこなわれる
- アプリはバックグラウンドでジョブやアラートを実行できなくなる
- アプリはプッシュ通知（Firebase Cloud Messaging を介して送信される優先度の高いメッセージなど）を受信できなくなる

休止状態はユーザーが次にアプリを操作した場合に終了となる。ただし、アプリが休止状態になる前にスケジュールされたジョブ、アラート、通知は、スケジュールを再設定が必要となる。

権限の自動リセットや自動休止機能で不都合が生じるようなアプリ、たとえばデバイスとサーバーの間で定期的にデータを同期するようなアプリにおいては、ユーザーはアプリ情報画面の「権限を削除して空き容量を増やす」オプションをオフにすることで、適用対象から除外することができる。

試験的にアプリを休止状態に移行させるには、以下のコマンドを実行する。こうすることで休止状態になった状況をシミュレートできる。

- 休止状態の動作を有効にする

```
$ adb shell device_config put app_hibernation app_hibernation_enabled true
```

- アプリを強制的に休止状態にする

```
$ adb shell cmd app_hibernation set-state org.jssec.android.activity.privateactivity true
```

5.5.3.9 パッケージアクセスの仕様変更にともなう API の戻り値の変化

デバイスに Android 12 がインストールされ、且つ、ターゲットに Android 11(API Level 30) 以降を指定する場合、パッケージアクセスの仕様変更にともない、以下のメソッドの戻り値はフィルタリングされた値となる。これは Android 11 においてなされた、パッケージアクセスの最小限の権限の原則に従ったものである。

- getAllPermissionGroups()

- getPermissionGroupInfo()
- getPermissionInfo()
- queryPermissionsByGroup()

動作確認として、以下のようなカスタムパーミッショングループを作成し、getAllPermissionGroups() の値がどのように変わるかを比較した。

```
<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"/>
```

- デバイスに Android 11 がインストールされ、ターゲットに Android 11(API Level 30) を指定した場合

```
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
android.permission-group.JSSEC
```

- デバイスに Android 12 がインストールされ、ターゲットに Android 11(API Level 30) を指定した場合

```
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
```

デバイスに Android 12 がインストールされ、且つ、ターゲットに Android 11(API Level 30) を指定した場合は、カスタムパーミッショングループの取得が出来ていないことがわかる。デバイスにインストールされたパッケージのリストもプライバシーという考えに基づく仕様であり、他のアプリにアクセスする必要がある場合は、<queries>により明示的に指定する必要がある。<queries>は本ガイドのいくつかのサンプルコードでも使用している。

5.5.3.10 Android 12 におけるマイクとカメラ

Android 12 では、クイック設定もしくはシステム設定の [プライバシー] 画面から、マイクとカメラのオン・オフが設定できる。この設定はデバイス上のすべてのアプリに対し反映される。

ユーザーが、マイクまたはカメラをオフにした設定した状態でそれらにアクセスするアプリを起動すると、システムからユーザーに対し、当該デバイスがオフになっていることが通知される。

この機能はサポートしているデバイスでのみ利用可能で、以下のコードにより確認できる。

```
SensorPrivacyManager sensorPrivacyManager = getApplicationContext().
↪getSystemService(SensorPrivacyManager.class);
boolean supportsMicrophoneToggle = sensorPrivacyManager.supportsSensorToggle(Sensors.
↪MICROPHONE);
boolean supportsCameraToggle = sensorPrivacyManager.supportsSensorToggle(Sensors.CAMERA);
```

マイクとカメラを利用するアプリにおいては、権限に沿って以下の方法によりアクセスすることが推奨される。

- ユーザーがアプリに CAMERA 権限を付与するまで、デバイスのカメラにアクセスしない。
- ユーザーがアプリに RECORD_AUDIO 権限を付与するまで、デバイスのマイクにアクセスしない。

アプリがマイクまたはカメラにアクセスすると、ステータスバーにアクセス中であることを示すアイコンが表示される。ユーザーは、クイック設定からアイコンをタップすることにより、現在どのアプリがマイクまたはカメラを使用しているのかを確認できる。

5.5.3.11 WebView における SameSite Cookie について

Android 12 をターゲットとするアプリの場合、WebView には以下のプライバシー保護の変更が適用される。

- SameSite 属性が設定されていない Cookie は、SameSite=Lax として扱われる。すなわちアクセスしているサイトと同じドメインの Cookie しか設定できない。
- SameSite=None の Cookie には、Secure 属性も明示的に指定する必要がある。すなわち HTTPS 接続を使わなければ Cookie の設定や読み取りができない。

これは、Cross Site Request Forgery (CSRF) 攻撃を防ぐためのものであり、Android 12 以降をターゲットとするアプリの WebView に適用される。

アプリで WebView を使用している場合、もしくは Cookie を使用するウェブサイトやサービスを管理している場合は、従前のフローが正しく動作するかを確認する必要がある。

なお、この変更は Android 12 における WebView バージョン（89.0.4385.0）以降から適用されるもので、バージョンの確認は、「設定」ー「アプリ」ー「Android System WebView」として表示される「アプリ情報」画面から行える[*25]。

[*25] Android 12(ビルド番号 SPB5.210812.002) の端末で確認したところ、WebView のバージョンは 93.0.4577.82

5.5.3.12 部分的画面共有

Android 15 では、新たに部分的画面共有機能が導入された。この機能により、一部のアプリウィンドウのみを共有または録画することが可能になる。この変更は、プライバシー保護や特定のアプリケーションのコンテンツにフォーカスするニーズに対応することを目的としている。以下に、実装方法の手順を示す。

1. オーバーレイ表示の許可を要求する

アプリが他のアプリの上にオーバーレイ表示を行うための許可を取得する必要がある。以下のコードで、オーバーレイ表示の許可をユーザーに要求する。

```
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()
        }
    }
}
```

上記のコードを実行すると、アプリケーションが他のアプリの上にオーバーレイ表示を行うための許可をユーザーに要求するダイアログが表示される。

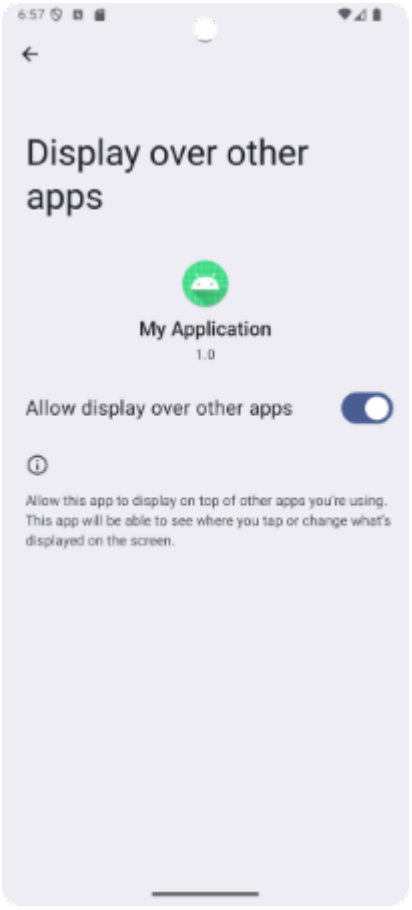

図 5.5.9　オーバーレイ表示を行うための許可

2. 画面キャプチャの許可を要求し、その結果を処理する

オーバーレイ表示の許可が付与されたら、次に MediaProjectionManager を使用して画面キャプチャの許可を要求し、その結果を処理する。

```
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)
        }
        if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.O) {
            startForegroundService(serviceIntent)
        } else {
            startService(serviceIntent)
        }
```

(次のページに続く)

(前のページからの続き)

```
    } else {
        Toast.makeText(this, "Permission denied", Toast.LENGTH_SHORT).show()
    }
}
```

上記のコードを実行すると、アプリケーションが他のアプリの画面キャプチャの許可をユーザーに要求するダイアログが表示される。

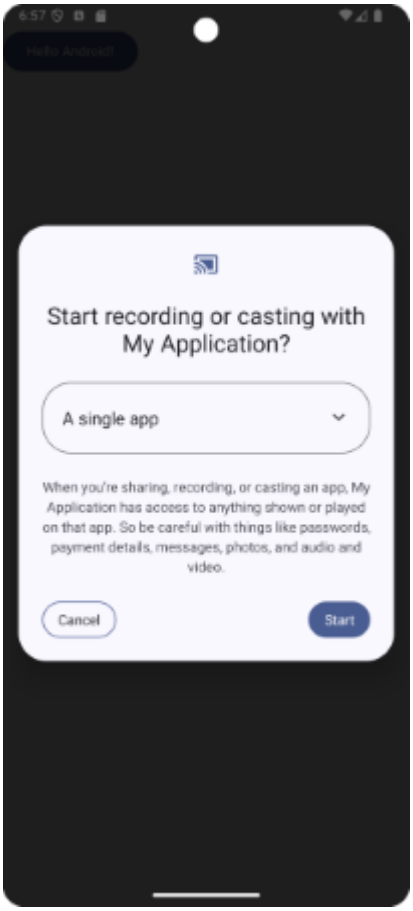

図 5.5.10　録画またはキャストすることを許可

3. キャプチャするアプリの選択

Start ボタンをクリックするとアプリの一覧が表示され、キャプチャするアプリを選択できるようになる。以下は Android 標準の Clock アプリを選択した場合。

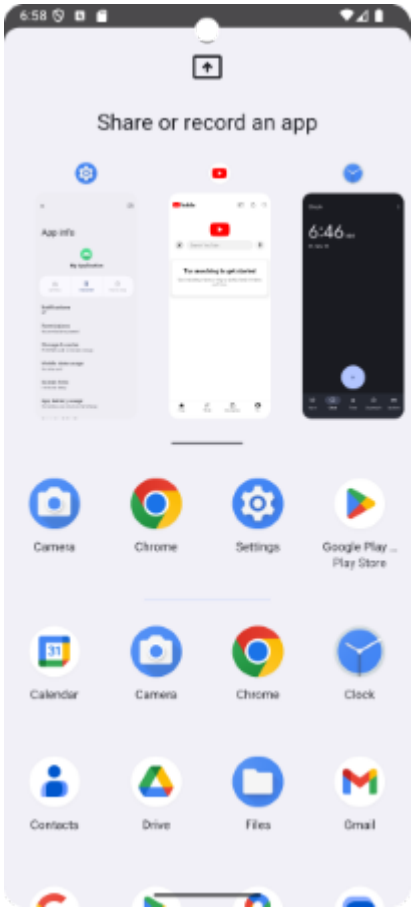

図 5.5.11　アプリの一覧

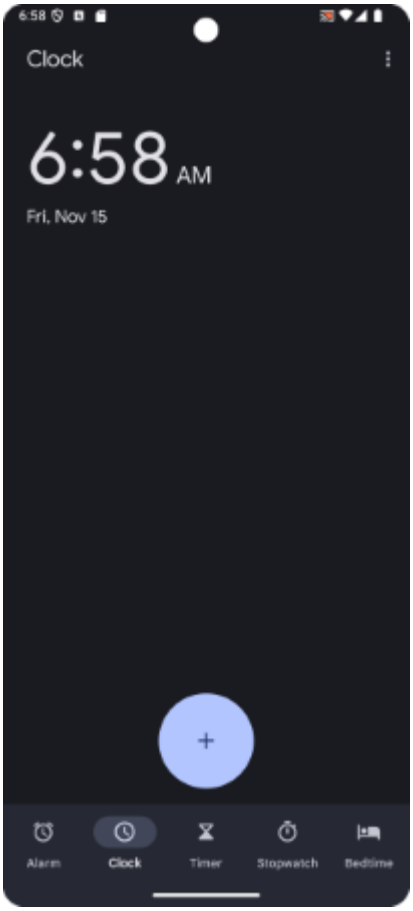

図 5.5.12　Clock アプリを選択

5.5.3.13 DND モードのグローバル状態管理の変更点

Android 14 以前では、setInterruptionFilter や setNotificationPolicy を使用して DND（Do Not Disturb）モードのグローバル状態やポリシーを直接変更できた。

```
fun setInterruptionFilter(context: Context, filter: Int) {
    val notificationManager = context.getSystemService(Context.NOTIFICATION_SERVICE) as␣
↪NotificationManager
    notificationManager.setInterruptionFilter(filter)
}
```

Android 15 以降では、アプリがデバイスの DND モードのグローバル状態やポリシーを直接変更することはできなくなっており、代わりに、AutomaticZenRule を使用することが推奨される。システムは、既存の中で最も制限の厳しいポリシーを優先するスキームで、これをグローバルポリシーに統合する。これは、悪意のあるアプリがユーザーの意図に反して DND モードを勝手にオンオフするのを防ぎ、ユーザーの設定や他のアプリの重要なルールを保護するためである。

AutomaticZenRule は、特定の条件が満たされたときに自動的に DND モードを有効または無効にするルールであり、これにより、DND モードを間接的に制御することができる。

ルールの作成

```
fun createAutomaticZenRule(context: Context) {
    val notificationManager = context.getSystemService(Context.NOTIFICATION_SERVICE) as␣
```

（次のページに続く）

(前のページからの続き)

```
↪NotificationManager

    // Define the condition for the rule (e.g., a condition to enable DND at a specific␣
↪time)
    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")
    }
}
```

ルールの更新

```
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,
```

(次のページに続く)

（前のページからの続き）

```
            zenRule.interruptionFilter,
            zenRule.isEnabled
        )

        notificationManager.updateAutomaticZenRule(ruleId, updatedZenRule)
        println("AutomaticZenRule updated: $ruleId")
    } else {
        println("AutomaticZenRule not found: $ruleId")
    }
}
```

グローバル状態に影響していた既存の API（setInterruptionFilter、setNotificationPolicy）を呼び出した場合、暗黙的に AutomaticZenRule が作成または更新される。この AutomaticZenRule は、API 呼び出しのサイクルに応じてオンまたはオフに切り替わる。

また、例えば setInterruptionFilter(NotificationManager.INTERRUPTION_FILTER_ALL) を使用して DND モードをオフにしようとする場合も暗黙的に AutomaticZenRule が作成され、システムが適切に管理する。

5.5.3.14 プライベート スペース

Android 15 では、プライベートスペース機能が実装された。プライベートスペースは、デバイス上に安全で隔離された環境を作成し、アプリやデータをプライベートに管理できる仕組みである。このスペースは端末のメインユーザーに紐づく形で作成され、利用やアプリのインストールはユーザーが決定する。そのため、アプリ開発者側で特別な API 対応や改修は原則不要だが、以下の仕様を把握しておくことが推奨される。

- プライベートスペースは Android のマルチユーザーモデルに基づき、独立したユーザープロファイルとして動作する。アプリはこのプロファイル内にインストールされ、データも分離される。ただし、暗号化などの追加保護が自動的に適用されるわけではないため、通常のアプリケーション同様の対策が必要となる。
- ストレージについては、プライベートスペース用の外部ストレージ領域が用意されており、メインユーザーとは異なる領域が割り当てられる。それぞれの領域のアプリは、原則として他の領域の外部ストレージに保存されたファイルへ直接アクセスできない。ロック解除時には、Sharesheet やシステムギャラリーピッカー経由でのみファイルのやり取りが可能となる。
- プライベートスペースのロック状態によって、アプリの動作は大きく異なる。ロック中は、プライベートスペース内のすべてのアプリが完全に停止し、通知の表示やバックグラウンド処理も行われない。また、通知やアプリの存在自体もランチャーや設定画面などから隠される。ロック解除後にのみ、通知やアプリが表示され、利用可能となる。
- このように、プライベートスペースはストレージ分離だけでなく、通知やアプリの可視性、バックグラウンド動作の制限など、ユーザーのプライバシー保護を多角的に実現している。特に、バックグラウンドでの動作や常時通知が必要なアプリ（例：医療系アプリ）については、プライベートスペース内での利用が推奨されない場合がある。
- メインスペースとプライベートスペースでは、異なる Google アカウントを利用することも可能である。

このように、プライベートスペースはストレージのみならず、通知やアプリの動作、可視性など多岐にわたる仕様を持つ。ストレージのみを意識した対策では不十分であり、アプリ開発者はプライベートスペースの全体像を理解したうえで、必要に応じた対応やユーザーへの案内を検討することが望ましい。

5.5.3.15 Contact Picker への移行と連絡先データの適切な取り扱い

Android 17 の連絡先プロバイダ（CP2）の厳格化に伴い、「連絡先はユーザーが選んだ分だけ共有する」モデルが推奨されている。これまで READ_CONTACTS 権限を利用して行われていた連絡先へのフルアクセスは大幅に制限され、Contact Picker によるユーザー選択型アクセスが基本となる。アプリは連絡先にフルアクセスする代わりに Contact Picker を利用し、ユーザーが個別に選択・許可したデータ（電話番号、メールアドレス、名前、住所、会社名・役職など）のみを取得する設計へ移行する必要がある。

この制限の背景には、アプリによる情報の過剰収集の見直しがある。従来の永続的なフルアクセスから、必要最小限の一時アクセスへと方針が転換された。通話アプリや連絡先管理アプリのように、どうしても連絡先へのフルアクセスが必要な場合は審査対象となり、Google Play Console にて正当な理由の申告が求められる。

また、Contact Picker で取得した情報は一時的な利用を前提とし、不要な長期間の保存やバックグラウンドでの継続的な利用は避けるべきである。ユーザーに明示することなく連絡先データを保存したりアップロードしたりする行為は、ユーザーデータポリシー違反とみなされる可能性がある。取得したデータは利用目的を限定し、データを保存せざるを得ない場合でも、その用途と範囲をユーザーに対して事前に明確に説明しなければならない。

ユーザー体験（UX）の観点からも、アプリ起動時にいきなり権限やデータの提供を要求する実装は避けるべきである。データを取得する前に、なぜそのデータが必要なのかを分かりやすく説明し、ユーザーの自発的な操作をトリガーとして Contact Picker を起動する設計にすることが、プライバシー保護とユーザーからの信頼獲得において重要である。

実装にあたっては、Android 17 以上の環境では既存の Intent.ACTION_PICK を用いた呼び出しも自動的に新しい Contact Picker のインターフェースへとアップグレードされる。しかし、複数のデータ種別を同時に指定するなど、新しい Contact Picker のフル機能を利用するためには、新設された Intent.ACTION_PICK_CONTACTS を明示的に使用することが推奨される。

```
// Android 17 以降で推奨される Contact Picker の起動例
val intent = Intent(Intent.ACTION_PICK_CONTACTS)

// ※必要に応じて、取得したいデータ種別（電話番号やメールアドレスなど）をエクストラで指定する
startActivityForResult(intent, REQUEST_CODE)
```

5.6 暗号技術を利用する

セキュリティの世界では、脅威を分析し対策するための観点として、機密性 (Confidentiality)、完全性 (Integrity)、可用性 (Availavility) という用語が使われる。それぞれ秘密のデータを第三者に見られないようにすること、参照するデータが改ざんされないように (または、改ざんされたことを検知) すること、サービスやデータが必要な時にいつでも利用できることを意味しており、セキュリティを確保する際に考慮すべき大切な要素である。中でも、機密性や完全性のために暗号技術が用いられることが多く、Android においてもアプリが機密性や完全性を実現するための様々な暗号機能が用意されている。

ここでは、Android アプリにおける暗号化・復号 (機密性の実現)、メッセージ認証コード/デジタル署名 (完全性の実現) の安全な実装方法をサンプルコードで示す。

5.6.1 サンプルコード

「暗号化・復号する (機密性を確保する)」、「改ざんを検知する (完全性を確認する)」というユースケースの具体的な用途や条件に対して、様々な暗号方式が開発されている。ここでは、暗号技術をどのような用途で利用するかという観点から、暗号方式を大きく 3 つに分類してサンプルコードを用意している。それぞれの暗号技術の特徴から利用すべき暗号方式・鍵の種類を判断することができる。より細やかな判断が必要な場合には、「5.6.3.1. 暗号方式の選択」も参照すること。

- 第三者の盗聴からデータを守る

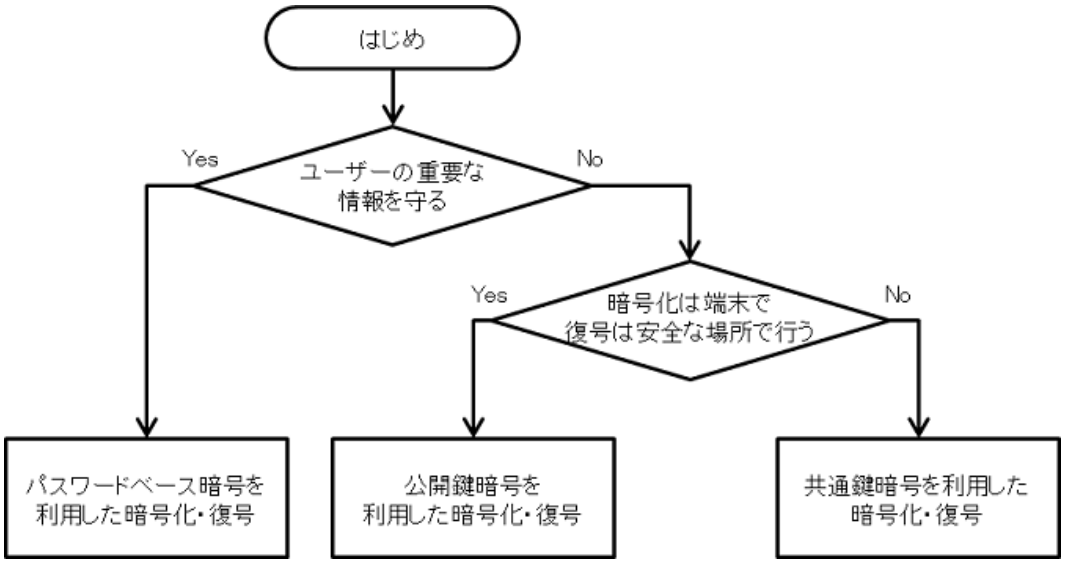

図 5.6.1　盗聴からデータを守るサンプルコードを選択するフローチャート

- 第三者によるデータの改ざんを検知する

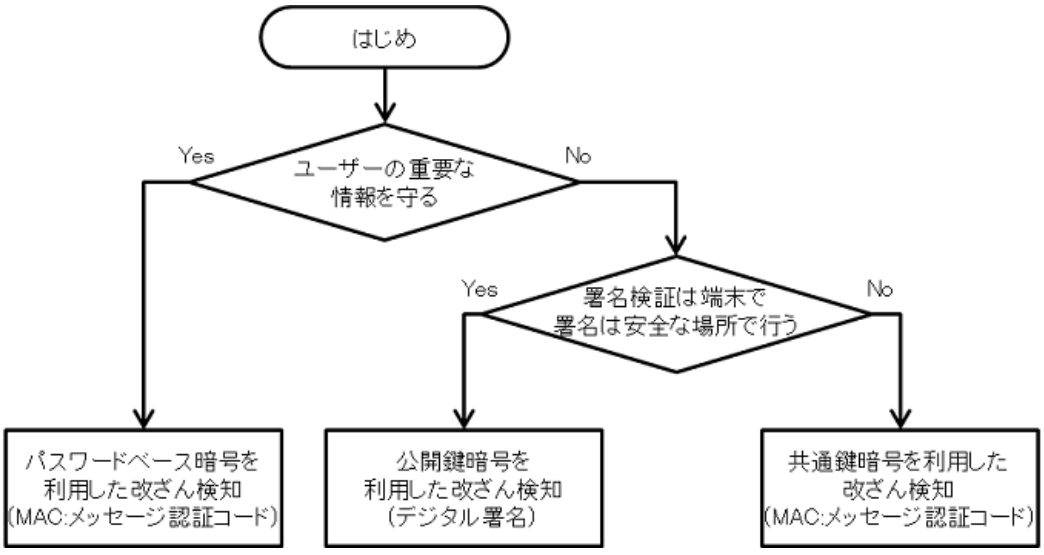

図 5.6.2　データの改ざんを検知するサンプルコードを選択するフローチャート

5.6.1.1 パスワード鍵を利用して暗号化・復号する

ユーザーの情報資産の秘匿性を守る目的にはパスワードベース鍵暗号を使うことができる。

ポイント：

1. 明示的に暗号モードとパディングを設定する
2. 脆弱でない (基準を満たす) 暗号技術（アルゴリズム・モード・パディング等）を使用する
3. パスワードから鍵を生成する場合は、Salt を使用する
4. パスワードから鍵を生成する場合は、適正なハッシュの繰り返し回数を指定する
5. 十分安全な長さを持つ鍵を利用する

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;

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;
```

(次のページに続く)

(前のページからの続き)

```
public final class AesCryptoPBEKey {

    // ★ポイント 1★ 明示的に暗号モードとパディングを設定する
    // ★ポイント 2★ 脆弱でない(基準を満たす)暗号技術(アルゴリズム・モード・パディング等)を使用する
    // Cipher クラスの getInstance に渡すパラメータ
    // (/[暗号アルゴリズム]/[ブロック暗号モード]/[パディングルール])
    // サンプルでは、暗号アルゴリズム=AES、ブロック暗号モード=CBC、パディングルール=PKCS7Padding
    private static final String TRANSFORMATION = "AES/CBC/PKCS7Padding";

    // 鍵を生成するクラスのインスタンスを取得するための文字列
    private static final String KEY_GENERATOR_MODE = "PBEWITHSHA256AND128BITAES-CBC-BC";

    // ★ポイント 3★ パスワードから鍵を生成する場合は、Salt を使用する
    // Salt のバイト長
    public static final int SALT_LENGTH_BYTES = 20;

    // ★ポイント 4★ パスワードから鍵を生成する場合は、適正なハッシュの繰り返し回数を指定する
    // PBE で鍵を生成する際の撹拌の繰り返し回数
    private static final int KEY_GEN_ITERATION_COUNT = 1024;

    // ★ポイント 5★ 十分安全な長さを持つ鍵を利用する
    // 鍵のビット長
    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;
    }

    AesCryptoPBEKey() {
        mIV = null;
        initSalt();
    }
```

(次のページに続く)

(前のページからの続き)

```
    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 {
            // ★ポイント 1★ 明示的に暗号モードとパディングを設定する
            // ★ポイント 2★ 脆弱でない (基準を満たす) 暗号技術（アルゴリズム・モード・パディング等）を
使用する
            Cipher cipher = Cipher.getInstance(TRANSFORMATION);

            // ★ポイント 3★ パスワードから鍵を生成する場合は、Salt を使用する
            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 {
            // ★ポイント 1★ 明示的に暗号モードとパディングを設定する
            // ★ポイント 2★ 脆弱でない (基準を満たす) 暗号技術（アルゴリズム・モード・パディング等）を
使用する
            Cipher cipher = Cipher.getInstance(TRANSFORMATION);

            // ★ポイント 3★ パスワードから鍵を生成する場合は、Salt を使用する
            SecretKey secretKey = generateKey(password, mSalt);
            IvParameterSpec ivParameterSpec = new IvParameterSpec(mIV);
```

(次のページに続く)

(前のページからの続き)

```
            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 char[] password, final byte[] salt) {
        SecretKey secretKey = null;
        PBEKeySpec keySpec = null;

        try {
            // ★ポイント 2★ 脆弱でない（基準を満たす）アルゴリズム・モード・パディングを使用する
            // 鍵を生成するクラスのインスタンスを取得する
            // 例では、AES-CBC 128 ビット用の鍵を SHA256 を利用して生成する KeyFactory を使用。
            SecretKeyFactory secretKeyFactory =
                    SecretKeyFactory.getInstance(KEY_GENERATOR_MODE);

            // ★ポイント 3★ パスワードから鍵を生成する場合は、Salt を使用する
            // ★ポイント 4★ パスワードから鍵を生成する場合は、適正なハッシュの繰り返し回数を指定する
            // ★ポイント 5★ 十分安全な長さを持つ鍵を利用する
            keySpec = new PBEKeySpec(password,
                    salt,
                    KEY_GEN_ITERATION_COUNT,
                    KEY_LENGTH_BITS);
            // password のクリア
            Arrays.fill(password, '?');
            // 鍵を生成する
            secretKey = secretKeyFactory.generateSecret(keySpec);
        } catch (NoSuchAlgorithmException e) {
        } catch (InvalidKeySpecException e) {
        } finally {
            keySpec.clearPassword();
        }

        return secretKey;
    }
```

(次のページに続く)

(前のページからの続き)

```
}
```

5.6.1.2 公開鍵を利用して暗号化・復号する

アプリ側では公開鍵を保持してデータの暗号化のみを行い、復号を異なる安全な場所 (サーバーなど) で秘密鍵によって行うような用途には公開鍵暗号を使うことができる。

ポイント：

1. 明示的に暗号モードとパディングを設定する
2. 脆弱でない (基準を満たす) 暗号技術（アルゴリズム・モード・パディング等）を使用する
3. 十分安全な長さを持つ鍵を利用する

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;
```

(次のページに続く)

(前のページからの続き)

```
import javax.crypto.IllegalBlockSizeException;
import javax.crypto.NoSuchPaddingException;

public final class RsaCryptoAsymmetricKey {

    // ★ポイント 1★ 明示的に暗号モードとパディングを設定する
    // ★ポイント 2★ 脆弱でない（基準を満たす）暗号技術（アルゴリズム・モード・パディング等）を使用する
    // Cipher クラスの getInstance に渡すパラメータ
    // （/[暗号アルゴリズム]/[ブロック暗号モード]/[パディングルール]）
    // サンプルでは、暗号アルゴリズム=RSA、ブロック暗号モード=NONE、パディングルール=OAEPPADDING
    private static final String TRANSFORMATION = "RSA/NONE/OAEPPADDING";

    // 暗号アルゴリズム
    private static final String KEY_ALGORITHM = "RSA";

    // ★ポイント 3★ 十分安全な長さを持つ鍵を利用する
    // 鍵長チェック
    private static final int MIN_KEY_LENGTH = 2000;

    RsaCryptoAsymmetricKey() {
    }

    public final byte[] encrypt(final byte[] plain, final byte[] keyData) {
        byte[] encrypted = null;

        try {
            // ★ポイント 1★ 明示的に暗号モードとパディングを設定する
            // ★ポイント 2★ 脆弱でない（基準を満たす）暗号技術（アルゴリズム・モード・パディング等）を
使用する
            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;
```

(次のページに続く)

(前のページからの続き)

```
    }

    public final byte[] decrypt(final byte[] encrypted, final byte[] keyData) {
        // 本来、復号処理はサーバー側で実装すべきものであるが、
        // 本サンプルでは動作確認用に、アプリ内でも復号処理を実装した。
        // 実際にサンプルコードを利用する場合は、アプリ内に秘密鍵を保持しないようにすること。

        byte[] plain = null;

        try {
            // ★ポイント 1★ 明示的に暗号モードとパディングを設定する
            // ★ポイント 2★ 脆弱でない（基準を満たす）暗号技術（アルゴリズム・モード・パディング等）を
使用する
            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 {
        }

        // ★ポイント 3★ 十分安全な長さを持つ鍵を利用する
        // 鍵長のチェック
```

(次のページに続く)

(前のページからの続き)

```
        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;
    }
}
```

5.6.1.3 共通鍵を利用して暗号化・復号する

サイズの大きなデータを扱う場合やアプリの資産・ユーザーの資産の秘匿性を守る目的で使うことができる。

ポイント：

1. 明示的に暗号モードとパディングを設定する
2. 脆弱でない (基準を満たす) 暗号技術（アルゴリズム・モード・パディング等）を使用する
3. 十分安全な長さを持つ鍵を利用する

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
```

(次のページに続く)

(前のページからの続き)

```
 *
 *      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 {

    // ★ポイント1★ 明示的に暗号モードとパディングを設定する
    // ★ポイント2★ 脆弱でない(基準を満たす)暗号技術（アルゴリズム・モード・パディング等）を使用する
    // Cipher クラスの getInstance に渡すパラメータ
    // （/[暗号アルゴリズム]/[ブロック暗号モード]/[パディングルール]）
    // サンプルでは、暗号アルゴリズム=AES、ブロック暗号モード=CBC、パディングルール=PKCS7Padding
    private static final String TRANSFORMATION = "AES/CBC/PKCS7Padding";

    // 暗号アルゴリズム
    private static final String KEY_ALGORITHM = "AES";

    // IVのバイト長
    public static final int IV_LENGTH_BYTES = 16;

    // ★ポイント3★ 十分安全な長さを持つ鍵を利用する
    // 鍵長チェック
    private static final int MIN_KEY_LENGTH_BYTES = 16;

    private byte[] mIV = null;

    public byte[] getIV() {
```

(次のページに続く)

(前のページからの続き)

```
        return mIV;
    }

    AesCryptoPreSharedKey(final byte[] iv) {
        mIV = iv;
    }

    AesCryptoPreSharedKey() {
    }

    public final byte[] encrypt(final byte[] keyData, final byte[] plain) {
        byte[] encrypted = null;

        try {
            // ★ポイント 1 ★ 明示的に暗号モードとパディングを設定する
            // ★ポイント 2 ★ 脆弱でない (基準を満たす) 暗号技術（アルゴリズム・モード・パディング等）を
使用する
            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;
    }

    public final byte[] decrypt(final byte[] keyData, final byte[] encrypted) {
        byte[] plain = null;

        try {
            // ★ポイント 1 ★ 明示的に暗号モードとパディングを設定する
            // ★ポイント 2 ★ 脆弱でない (基準を満たす) 暗号技術（アルゴリズム・モード・パディング等）を
使用する
            Cipher cipher = Cipher.getInstance(TRANSFORMATION);
```

(次のページに続く)

(前のページからの続き)

```
            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 {
            // ★ポイント 3★ 十分安全な長さを持つ鍵を利用する
            if (keyData.length >= MIN_KEY_LENGTH_BYTES) {
                // ★ポイント 2★ 脆弱でない (基準を満たす) 暗号技術 (アルゴリズム・モード・パディング等)
を使用する
                secretKey = new SecretKeySpec(keyData, KEY_ALGORITHM);
            }
        } catch (IllegalArgumentException e) {
        } finally {
        }

        return secretKey;
    }
}
```

5.6.1.4 パスワード鍵を利用して改ざんを検知する

ユーザーの情報資産の完全性をチェックする目的にはパスワードベース (共通鍵) 暗号を使うことができる。

ポイント：

1. 明示的に暗号モードとパディングを設定する

2. 脆弱でない (基準を満たす) 暗号技術（アルゴリズム・モード・パディング等）を使用する
3. パスワードから鍵を生成する場合は、Salt を使用する
4. パスワードから鍵を生成する場合は、適正なハッシュの繰り返し回数を指定する
5. 十分安全な長さを持つ鍵を利用する

HmacPBEKey.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.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 {

    // ★ポイント 1★ 明示的に暗号モードとパディングを設定する
    // ★ポイント 2★ 脆弱でない (基準を満たす) 暗号技術（アルゴリズム・モード・パディング等）を使用する
    // Mac クラスの getInstance に渡すパラメータ （認証モード）
    private static final String TRANSFORMATION = "PBEWITHHMACSHA1";

    // 鍵を生成するクラスのインスタンスを取得するための文字列
    private static final String KEY_GENERATOR_MODE = "PBEWITHHMACSHA1";
```

(次のページに続く)

(前のページからの続き)

```
    // ★ポイント 3★ パスワードから鍵を生成する場合は、Salt を使用する
    // Salt のバイト長
    public static final int SALT_LENGTH_BYTES = 20;

    // ★ポイント 4★ パスワードから鍵を生成する場合は、適正なハッシュの繰り返し回数を指定する
    // PBE で鍵を生成する際の撹拌の繰り返し回数
    private static final int KEY_GEN_ITERATION_COUNT = 1024;

    // ★ポイント 5★ 十分安全な長さを持つ鍵を利用する
    // 鍵のビット長
    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 {
            // ★ポイント 1★ 明示的に暗号モードとパディングを設定する
            // ★ポイント 2★ 脆弱でない (基準を満たす) 暗号技術（アルゴリズム・モード・パディング等）を
使用する
            Mac mac = Mac.getInstance(TRANSFORMATION);
```

(次のページに続く)

(前のページからの続き)

```
            // ★ポイント 3★ パスワードから鍵を生成する場合は、Salt を使用する
            SecretKey secretKey = generateKey(password, mSalt);
            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 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 {
            // ★ポイント 2★ 脆弱でない(基準を満たす)アルゴリズム・モード・パディングを使用する
            // 鍵を生成するクラスのインスタンスを取得する
            // 例では、AES-CBC 128 ビット用の鍵を SHA1 を利用して生成する KeyFactory を使用。
            SecretKeyFactory secretKeyFactory =
                SecretKeyFactory.getInstance(KEY_GENERATOR_MODE);

            // ★ポイント 3★ パスワードから鍵を生成する場合は、Salt を使用する
            // ★ポイント 4★ パスワードから鍵を生成する場合は、適正なハッシュの繰り返し回数を指定する
            // ★ポイント 5★ 十分安全な長さを持つ鍵を利用する
            keySpec =
                new PBEKeySpec(password, salt, KEY_GEN_ITERATION_COUNT, KEY_LENGTH_BITS);
            // password のクリア
            Arrays.fill(password, '?');
            // 鍵を生成する
            secretKey = secretKeyFactory.generateSecret(keySpec);
```

(次のページに続く)

(前のページからの続き)

```
        } catch (NoSuchAlgorithmException e) {
        } catch (InvalidKeySpecException e) {
        } finally {
            keySpec.clearPassword();
        }

        return secretKey;
    }

}
```

5.6.1.5 公開鍵を利用して改ざんを検知する

異なる安全な場所 (サーバーなど) で秘密鍵による署名を行ったデータに対して、アプリ側では公開鍵を保持してデータの署名検証のみを行うような用途には公開鍵暗号を使うことができる。

ポイント：

1. 明示的に暗号モードとパディングを設定する
2. 脆弱でない (基準を満たす) 暗号技術（アルゴリズム・モード・パディング等）を使用する
3. 十分安全な長さを持つ鍵を利用する

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;
```

(次のページに続く)

(前のページからの続き)

```
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 {

    // ★ポイント 1★ 明示的に暗号モードとパディングを設定する
    // ★ポイント 2★ 脆弱でない (基準を満たす) 暗号技術（アルゴリズム・モード・パディング等）を使用する
    // Cipher クラスの getInstance に渡すパラメータ
    // （/[暗号アルゴリズム]/[ブロック暗号モード]/[パディングルール])
    // サンプルでは、暗号アルゴリズム=RSA、ブロック暗号モード=NONE、パディングルール=OAEPPADDING
    private static final String TRANSFORMATION = "SHA256withRSA";

    // 暗号アルゴリズム
    private static final String KEY_ALGORITHM = "RSA";

    // ★ポイント 3★ 十分安全な長さを持つ鍵を利用する
    // 鍵長チェック
    private static final int MIN_KEY_LENGTH = 2000;

    RsaSignAsymmetricKey() {
    }

    public final byte[] sign(final byte[] plain, final byte[] keyData) {
        // 本来、署名処理はサーバー側で実装すべきものであるが、
        // 本サンプルでは動作確認用に、アプリ内でも署名処理を実装した。
        // 実際にサンプルコードを利用する場合は、アプリ内に秘密鍵を保持しないようにすること。

        byte[] sign = null;

        try {
            // ★ポイント 1★ 明示的に暗号モードとパディングを設定する
            // ★ポイント 2★ 脆弱でない (基準を満たす) 暗号技術（アルゴリズム・モード・パディング等）を
使用する
            Signature signature = Signature.getInstance(TRANSFORMATION);

            PrivateKey privateKey = generatePriKey(keyData);
            signature.initSign(privateKey);
            signature.update(plain);
```

(次のページに続く)

(前のページからの続き)

```
            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 {
            // ★ポイント 1 ★ 明示的に暗号モードとパディングを設定する
            // ★ポイント 2 ★ 脆弱でない (基準を満たす) 暗号技術（アルゴリズム・モード・パディング等）を
使用する
            Signature signature = Signature.getInstance(TRANSFORMATION);

            PublicKey publicKey = generatePubKey(keyData);
            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) {
```

(次のページに続く)

(前のページからの続き)

```
        } catch (NoSuchAlgorithmException e) {
        } catch (InvalidKeySpecException e) {
        } finally {
        }

        // ★ポイント 3★ 十分安全な長さを持つ鍵を利用する
        // 鍵長のチェック
        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;
    }
}
```

5.6.1.6 共通鍵を利用して改ざんを検知する

アプリの資産・ユーザーの資産の完全性をチェックする目的で使うことができる。

ポイント：

1. 明示的に暗号モードとパディングを設定する
2. 脆弱でない (基準を満たす) 暗号技術（アルゴリズム・モード・パディング等）を使用する
3. 十分安全な長さを持つ鍵を利用する

HmacPreSharedKey.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.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 {

    // ★ポイント 1★ 明示的に暗号モードとパディングを設定する
    // ★ポイント 2★ 脆弱でない（基準を満たす）暗号技術（アルゴリズム・モード・パディング等）を使用する
    // Mac クラスの getInstance に渡すパラメータ （認証モード）
    private static final String TRANSFORMATION = "HmacSHA256";

    // 暗号アルゴリズム
    private static final String KEY_ALGORITHM = "HmacSHA256";

    // ★ポイント 3★ 十分安全な長さを持つ鍵を利用する
    // 鍵長チェック
    private static final int MIN_KEY_LENGTH_BYTES = 16;

    HmacPreSharedKey() {
    }

    public final byte[] sign(final byte[] plain, final byte[] keyData) {
```

（次のページに続く）

(前のページからの続き)

```
        return calculate(plain, keyData);
    }

    public final byte[] calculate(final byte[] plain, final byte[] keyData) {
        byte[] hmac = null;

        try {
            // ★ポイント 1 ★ 明示的に暗号モードとパディングを設定する
            // ★ポイント 2 ★ 脆弱でない (基準を満たす) 暗号技術（アルゴリズム・モード・パディング等）を
使用する
            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 {
            // ★ポイント 3 ★ 十分安全な長さを持つ鍵を利用する
            if (keyData.length >= MIN_KEY_LENGTH_BYTES) {
                // ★ポイント 2 ★ 脆弱でない (基準を満たす) 暗号技術（アルゴリズム・モード・パディング等）
を使用する
```

(次のページに続く)

(前のページからの続き)

```
                secretKey = new SecretKeySpec(keyData, KEY_ALGORITHM);
            }
        } catch (IllegalArgumentException e) {
        } finally {
        }

        return secretKey;
    }
}
```

5.6.2 ルールブック

暗号技術を利用する際には以下のルールを守ること。

1. 暗号を指定する場合は、明示的に暗号モードとパディングを設定する （必須）
2. 脆弱でないアルゴリズム（基準を満たすもの）を使用する （必須）
3. パスワードベース暗号のパスワードを端末内に保存しないこと （必須）
4. パスワードから鍵を生成する場合は、*Salt* を使用する （必須）
5. パスワードから鍵を生成する場合は、適正なハッシュの繰り返し回数を指定する （必須）
6. パスワードの強度を高める工夫をする （推奨）

5.6.2.1 暗号を指定する場合は、明示的に暗号モードとパディングを設定する （必須）

暗号化やデータ検証などで暗号技術を利用する場合、明示的に暗号モードとパディングを設定すること。Android アプリの開発で暗号技術を利用する場合、主に java.crypto のクラスである Cipher クラスを利用する。Cipher クラスを利用する際、初めにどのような暗号を利用するかを指定して Cipher クラスのオブジェクトを生成する。この指定を Transformation と呼ぶが、Transformation の指定の書式には以下の 2 種類が存在する

- ”algorithm/mode/padding”
- ”algorithm”

後者の場合、暗号モードとパディングは、Android で利用可能な暗号化サービスプロバイダ固有のデフォルト値が使用される。このデフォルト値は、利便性や互換性などを優先し、あまり安全でないものが設定されていることがあり、セキュリティを確保するためには必ず、前者の暗号モードとパディングを指定する書式を利用する必要がある。

5.6.2.2 脆弱でないアルゴリズム（基準を満たすもの）を使用する （必須）

暗号技術を利用する場合、アルゴリズムとして脆弱でない、一定の基準を満たしたものを使用すること。また、アルゴリズムとして複数の鍵長を許容する場合、アプリの製品ライフタイムを考慮して十分安全な鍵長を利用すること。さらには、暗号モードやパディングモードについても既知の攻撃方法が存在するものもあり、脆弱でない十分安全な方式を利用すること。

脆弱な暗号技術を利用した場合、例えば第三者からの盗聴を防ぐために暗号化したファイルでも、有効な保護になっておらず、第三者の盗聴を許してしまう為である。IT 技術の進化に伴い、暗号解析の技術も向上されるため、利用する技術はアプリが稼働することを想定する期間中、安全が保護されることが期待できるアルゴリズムを検討し選択する必要がある。

アルゴリズムのライフタイム（特定の暗号化アルゴリズムによって保護されたデータが安全なままである推定期間）や暗号技術に関して、以下に示す各国の基準を参考にすること。

表 5.6.1　NIST, FIPS（米国）[*26][*27][*28]

アルゴリズムのライフタイム	共通鍵暗号	公開鍵暗号（例：**RSA, DSA, DH**）	楕円曲線暗号（例：**ECDSA)**	ハッシュ値（デジタル署名、ハッシュ関数）	ハッシュ値（**HMAC**、鍵導出関数、乱数生成器）
レガシー（最低暗号強度 80bit 以下）	80	1024	160	160	-
～2030 年（最低暗号強度 112bit 以上）	112	2048	224	224	-
2030 年～（最低暗号強度 128bit 以上）	128	3072	256	256	160

単位：bit

表 5.6.2　ECRYPT II (EU)[*29]

アルゴリズムのライフタイム	共通鍵暗号	公開鍵暗号	楕円曲線暗号	ハッシュ値
保護期間 4 年以内	80	1248	160	160
保護期間 10 年	96	1776	192	192
保護期間 20 年	112	2432	224	224
保護期間 30 年	128	3248	256	256
量子コンピュータに対する防御	256	15424	512	512

単位：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)

表 5.6.3　CRYPTREC（日本） 電子政府推奨暗号リスト[*30]

技術分類		名称
公開鍵暗号	署名	DSA,ECDSA,RSA=PSS,RSASSA=PKCS1=V1_5
	守秘	RSA-OAEP
	鍵共有	DH,ECDH
共通鍵暗号	64 ビットブロック暗号	3-key Triple DES
	128 ビットブロック暗号	AES,Camellia
	ストリーム暗号	KCipher-2
ハッシュ関数		SHA-256,SHA-384,SHA-512
暗号利用モード	秘匿モード	CBC,CFB,CTR,OFB
	認証付き秘匿モード	CCM,GCM
メッセージ認証コード		CMAC,HMAC
エンティティ認証		ISO/IEC 9798-2,ISO/IEC 9798-3

5.6.2.3 パスワードベース暗号のパスワードを端末内に保存しないこと （必須）

パスワードベース暗号で、ユーザーから入力を受けたパスワードをベースに暗号鍵を生成する場合、入力されたパスワードを端末内に保存しないこと。パスワードベース暗号の利点は暗号鍵の管理を不要とすることであり、端末内にパスワードを保存してしまうと、その利点が得られなくなるためである。もちろん、端末内にパスワードを保存することで、他のアプリから盗聴されてしまうリスクが生じることもあり、安全面から考えても端末内への保存はしてはならない。

5.6.2.4 パスワードから鍵を生成する場合は、Salt を使用する （必須）

パスワードベース暗号で、ユーザーから入力を受けたパスワードをベースに暗号鍵を生成する場合、必ず Salt を使用すること。また、端末内の別ユーザーに機能を提供する場合は、ユーザー毎に異なる Salt を利用すること。Salt を利用せずに単純なハッシュ関数のみで暗号鍵を生成した場合、レインボーテーブルと呼ばれる技術を使うことで、容易に元のパスワードを類推することができるためである。Salt を付けた場合、同じパスワードから生成した鍵であっても異なる鍵（ハッシュ値）が生成されるようになり、レインボーテーブルによる鍵の探索を妨害することができる。

パスワードから鍵を生成する場合に、Salt を使用する例

```
public final byte[] encrypt(final byte[] plain, final char[] password) {
    byte[] encrypted = null;

    try {
        // ★ポイント★ 明示的に暗号モードとパディングを設定する
        // ★ポイント★ 脆弱でない（基準を満たす）暗号技術（アルゴリズム・モード・パディング等）を使用する
        Cipher cipher = Cipher.getInstance(TRANSFORMATION);

        // ★ポイント★ パスワードから鍵を生成する場合は、Salt を使用する
        SecretKey secretKey = generateKey(password, mSalt);
```

[*30] https://www.cryptrec.go.jp/list.html

5.6.2.5 パスワードから鍵を生成する場合は、適正なハッシュの繰り返し回数を指定する （必須）

パスワードベース暗号で、ユーザーから入力を受けたパスワードをベースに暗号鍵を生成する場合、鍵生成で適用するハッシュ処理を十分安全な回数繰り返す（ストレッチングする）こと。一般に 1,000 回以上の繰り返しであれば良いとされる。さらに重要な資産を保護するために利用する場合は 1,000,000 回以上の繰り返しを行うこと。ハッシュ関数は一回の計算にかかる時間が非常に短時間であるため、攻撃者による総当たり攻撃が容易になる原因にもなっている。そのため、ハッシュ処理を繰り返し用いるストレッチング手法で、わざと時間がかかるようにして総当たり攻撃を困難にする。なお、ストレッチング回数はアプリの処理速度にも影響を及ぼすため、設定する回数には注意すること。

パスワードから鍵を生成する場合に、ハッシュの繰り返し回数を指定する例

```
private static final SecretKey generateKey(final char[] password, final byte[] salt) {
    SecretKey secretKey = null;
    PBEKeySpec keySpec = null;

    // ～省略～

    // ★ポイント★ パスワードから鍵を生成する場合は、Salt を使用する
    // ★ポイント★ パスワードから鍵を生成する場合は、適正なハッシュの繰り返し回数を指定する
    // ★ポイント★ 十分安全な長さを持つ鍵を利用する
    keySpec = new PBEKeySpec(password,
                             salt,
                             KEY_GEN_ITERATION_COUNT,
                             KEY_LENGTH_BITS);
```

5.6.2.6 パスワードの強度を高める工夫をする （推奨）

パスワードベース暗号で、ユーザーから入力を受けたパスワードをベースに暗号鍵を生成する場合、暗号鍵の強度はユーザーのパスワードの強度に直結するため、ユーザーから受け付けるパスワードの強度を高める工夫をすることが望ましい。例えば、パスワードの最低長を 8 文字以上としたり、複数の文字種の利用を必須としたり (英数記号を各 1 文字以上など) するなどの方法が考えられる。

5.6.3 アドバンスト

5.6.3.1 暗号方式の選択

「サンプルコード」では、暗号化・復号、改ざん検知の各々について 3 つの暗号方式の実装例を示した。用途によってどの暗号方式を使用するかも、「図 5.6.1 盗聴からデータを守るサンプルコードを選択するフローチャート」、「図 5.6.2 データの改ざんを検知するサンプルコードを選択するフローチャート」を利用して大まかに判断することができる。一方で、暗号方式の選択は、その特徴を比較することでより細やかに判断することもできる。以下に比較を示す。

暗号化・復号における暗号方式の比較

公開鍵暗号は、処理コストが高いため大きなサイズのデータ処理には向かないが、暗号化と復号に使う鍵が異なるため、公開鍵のみアプリ側で扱い（暗号化のみ行い）、復号を別の（安全な）場所で行うような場合は鍵の管理が比較的容易である。共通鍵暗号は、制限の少ない万能な暗号方式であるが、暗号化・復号ともに同じ鍵を使用するため、アプリ内に鍵を安全に保持する必要があり、鍵の保護が難しい。パスワードベース暗号 (パスワードを元にした共通鍵暗号) は、ユーザーのパスワードから鍵を生成するため、端末内に鍵に関する秘密を保存する必要がない。用途としては、ユーザーの資

産を保護する場合のみに使える。また、暗号の強度がパスワードの強度に依存するため、資産の保護レベルに応じてパスワードを複雑にする必要がある。「5.6.2.6. パスワードの強度を高める工夫をする （推奨)」も参照すること。

表 5.6.4　暗号化・復号における暗号方式の比較

	公開鍵	共通鍵	パスワードベース
サイズの大きなデータの処理	NG(処理コストが高い)	OK	OK
アプリ (サービス) 資産の保護	OK	OK	NG(ユーザーによる盗聴が可能)
ユーザー資産の保護	OK	OK	OK
暗号の強度	鍵の長さに依存	鍵の長さに依存	パスワードの強度、Salt、ハッシュの繰り返し回数に依存
鍵の保護	容易 (公開鍵のみ)	困難	容易
アプリで行う処理	暗号化 (復号はサーバーなど)	暗号化・復号	暗号化・復号

改ざん検知における暗号方式の比較

データのサイズの項を除いた以外は、暗号化・復号における暗号方式の比較とほぼ同様である。

表 5.6.5　改ざん検知における暗号方式の比較

	公開鍵	共通鍵	パスワードベース
アプリ (サービス) 資産の保護	OK	OK	NG(ユーザーによる改ざんが可能)
ユーザー資産の保護	OK	OK	OK
暗号の強度	鍵の長さに依存	鍵の長さに依存	パスワードの強度、Salt、ハッシュの繰り返し回数に依存
鍵の保護	容易 (公開鍵のみ)	困難 (「5.6.3.3. 鍵の保護」参照)	容易
アプリで行う処理	署名検証 (署名はサーバーなど)	MAC 計算・MAC 検証	MAC 計算・MAC 検証

MAC：メッセージ認証コード

なお本ガイドでは、「3.1.3. 資産分類と保護施策」における資産レベル低中位の資産を主な保護対象としている。暗号の使用は鍵の管理問題など一般の保護施策 (アクセス制御など) と比べて多くの検討事項を伴うため、Android OS のセキュリティモデルでは資産の保護を充分にできない場合のみ、暗号の使用を検討することが望ましい。

5.6.3.2 乱数の生成

暗号技術の利用において、強固な暗号アルゴリズムや暗号モードを選択し、十分な長さの鍵を使用することは、アプリやサービスで扱うデータのセキュリティを確保する上で非常に重要である。しかし、これらの選択が適切であっても秘密の起点となる鍵が漏洩したり、予測されたりすると、アルゴリズム等で保証するセキュリティの強度が全く意味をなさなくなってしまう。また、AES 等の共通暗号で使用する初期化ベクトル (IV) やパスワードベース暗号で使用する Salt(ソ

ルト) に関しても、偏りが大きいと第三者による攻撃が容易になり、情報の漏洩や改ざんなどの被害に繋がる可能性が高くなる。このような事態を防ぐには、第三者に鍵や IV の値の予測が困難な方法で生成する必要があり、それを実現するために極めて重要な役割を果たすのが乱数である。乱数を生成する装置は乱数生成器と呼ばれ、センサー等を用いて予測・再現が不可能とされる自然状態を観測することで乱数を生成するハードウェアの乱数生成器 (RNG) に対して、ソフトウェアで実現する乱数生成器を疑似乱数生成器 (PRNG) と呼ぶのが一般的である。

Android アプリでは、暗号用途での利用に対して十分セキュアな乱数を SecureRandom クラス経由で取得することができる。SecureRandom クラスは、その機能を提供する Provider と呼ばれる実装を内部に複数持つことが可能であり、Provider を明示的に指定しない場合はデフォルトの Provider が選ばれる。暗号学的に安全ではない[*31] SHA1PRNG アルゴリズムを提供する Crypto Provider は、Android 7.0(API Level 24) で非推奨となり、Android 9.0(API Level 28) で削除された[*32][*33][*34]。Crypto Provider を指定して SecureRandom を使うと、Android 9.0 以降の端末では常に、Android 7.0 以降の端末でも targetSdkVersion>=24 であれば、NoSuchProviderException が発生する。そのため、原則として Provider を指定しないで SecureRandom を使うことを推奨する。以下に SecureRandom の使い方の例を示す。

SecureRandom の使用 (デフォルトの実装を使用する)

```
import java.security.SecureRandom;
// ～省略～
    SecureRandom random = new SecureRandom();
    byte[] randomBuf = new byte [128];

    random.nextBytes(randomBuf);
// ～省略～
```

SecureRandom のようなプログラムで実現されている疑似乱数生成器は一般に「図 5.6.3 疑似乱数生成器の内部プロセス」のように動作しており、乱数の種を入力して内部状態を初期化すると、乱数を生成する度に内部状態を一定のアルゴリズムで更新することで、次々と乱数列の生成が可能になる。

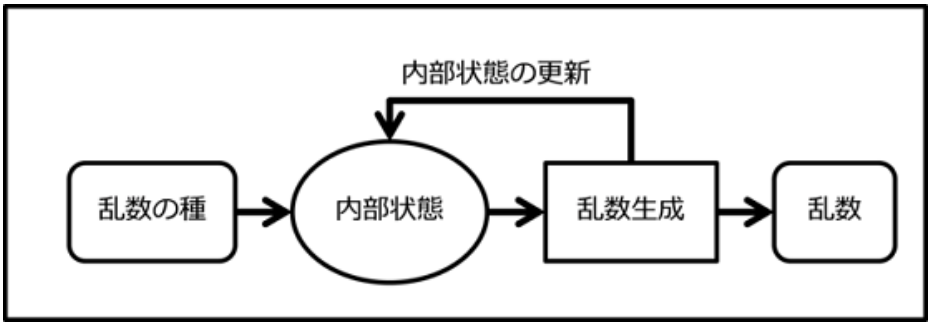

図 5.6.3　疑似乱数生成器の内部プロセス

乱数の種

疑似乱数生成器において乱数の種の役割は極めて重要である。

既に説明したように疑似乱数生成器は乱数の種による初期化が必要である。また、乱数の種で初期化された後は決められたアルゴリズムで乱数の生成が行われるので、同じ乱数の種からは同じ乱数列が生成されることになる。これは乱数

[*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)

の種が第三者に知られ (盗聴され) たり、予測可能であったりすると、第三者が同じ乱数列を取得することになり、乱数を起点にした機密性や完全性が失われることを意味している。

そのため、乱数の種はそれ自身機密性の高い情報であり、かつ予測困難でなければならない。例えば時刻情報や端末固有値 (MAC アドレス、IMEI、Android ID など) を乱数の種として使うべきではない。多くの Android 端末においては、/dev/urandom, /dev/random 等が利用可能であり、デフォルトで提供される SecureRandom の実装もそれらのデバイスファイルを使って乱数の種を設定している。また、機密性に関しては、乱数の種がメモリ内にのみ存在するならば、root 権限を取得したマルウェア・ツールでない限り、盗聴の危険性は低い。仮に、root 化された端末でも安全にする必要の場合は、セキュア設計・実装の専門家と相談して対応すること。

疑似乱数生成器の内部状態

疑似乱数生成器の内部状態は、乱数の種によって初期化され、乱数を生成する毎に状態が更新される。また、乱数の種と同様に、同じ内部状態の疑似乱数生成器が存在した場合は、その後に生成される乱数列はどちらもまったく同じものになる。よって、内部状態も第三者に対して、盗聴されないように気を付けなければならない。ただし、内部状態はメモリ内に存在するため、root 権限を取得したマルウェア・ツールでない限り、盗聴の危険性は低い。root 化された端末でも安全にする必要の場合は、セキュア設計・実装の専門家と相談して対応すること。

5.6.3.3 鍵の保護

センシティブな情報の安全性 (機密性、完全性) を実現するために暗号技術を利用する際に、堅牢な暗号化アルゴリズムや鍵長を選んでも、鍵そのもののデータが簡単に取得できる状態では、第三者から情報の安全性を保つことはできない。そのため、暗号技術を利用する際の鍵の取り扱いは、もっとも重要な検討項目の一つである。一方で、守るべき資産のレベルによっては鍵の取り扱いは非常に高度な設計・実装を必要とするため本ガイドの範囲を超える。そこで本ガイドでは、鍵の配置場所と用途ごとに安全な鍵の取り扱いをするための基本的な考え方を示すにとどめ、具体的な実現方法に関しては言及しない。必要に応じて Android セキュリティ設計・実装に詳しい専門家に相談することをお勧めする。

まずは、Android スマートフォン/タブレットにおいて暗号化等に使う鍵が存在する可能性のある場所とその保護方針の概観を「図 5.6.4 暗号鍵が存在する場所と保護方針」に示す。

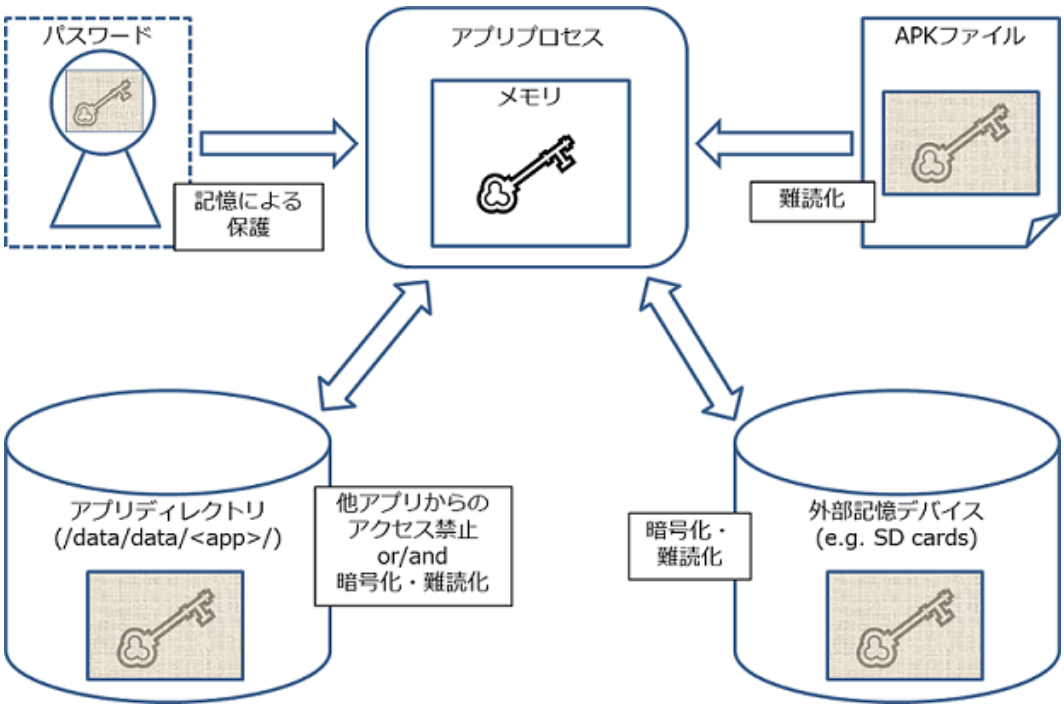

図 5.6.4　暗号鍵が存在する場所と保護方針

また、鍵によって守る資産の資産分類やオーナーによる保護方針の違いを表にまとめておく。資産レベルについては「3.1.3. 資産分類と保護施策」を参照すること。

表 5.6.6　資産分類と保護方針-1

資産のオーナー	端末のユーザー		アプリ・サービス提供者	
資産レベル	高位	中低位	高位	中低位
鍵の保存場所	保護方針			
ユーザーの記憶	パスワードの強度の向上		ユーザーパスワードの利用不可	
アプリディレクトリ (非公開ストレージ)	鍵データの暗号化･難読化	アプリ以外からの読み書き禁止	鍵データの暗号化･難読化	アプリ以外からの読み書き禁止

鍵の保存場所が APK ファイルや SD カードなどの公開ストレージの場合は次のようになる。

表 5.6.7　資産分類と保護方針-2

鍵の保存場所	保護方針
APK ファイル	鍵データの難読化 (Proguard など Java の難読化ツールの多くは、データ (文字) 列を難読化しないことに注意)
SD カードなど (公開ストレージ)	鍵データの暗号化・難読化

以下、鍵の存在場所ごとに保護方針の補足を示す。

ユーザーの記憶

この項ではパスワードベース暗号を想定している。パスワードから生成される鍵の場合は、鍵の保管場所はユーザーの記憶の中にあるため、保存場所からマルウェア等に漏洩することはない。ただし、パスワードの強度によっては簡単に鍵が再現されてしまうため、サービスへのログインパスワード等と同様にユーザーに対して一定のパスワード強度になるように UI によって制限したり、メッセージによる注意喚起したりすることが必要となる。「5.5.2.6. アプリ・プライバシーポリシー概要版を *assets* フォルダ内に配置しておく （推奨)」も参照すること。また、ユーザーの記憶にあるパスワードは忘れる可能性も考慮にいれておかなければならない。パスワードを忘れた場合でもデータの復元が必要な場合は、バックアップデータを端末以外の安全な場所 (サーバーなど) に保存しておく必要がある。

アプリディレクトリ内

アプリのディレクトリ内に Private モードで保存した場合は他のアプリから鍵データを読み取られることはない。また、バックアップ機能をアプリで無効にした場合には、ユーザーも参照できなくなるので、アプリの資産を守る鍵をアプリディレクトリ内に保存する場合はバックアップを無効にすれば良い。

ただし、root 権限を持つアプリやユーザーからも鍵を保護したい場合は、鍵データを暗号化もしくは難読化する必要がある。ユーザー資産を守る鍵を暗号化する場合にはパスワードベース暗号が利用できる。ユーザーにも秘密にしたいアプリの資産を暗号化する鍵を保護する場合には、APK ファイルに鍵暗号化鍵を埋める必要があり、鍵暗号化鍵データを難読化する必要がある。

APK ファイル内

APK ファイル内に存在するデータは読み取り可能であるため、基本的には鍵のような秘匿データを含めるべきではない。鍵を含める場合は鍵データを難読化して、容易に APK ファイルから鍵データが取得されないように対策を講じる必要がある。

公開ストレージ **(SD** カードなど) 内

公開ストレージはすべてのアプリから読み取り可能なため、基本的には鍵のような秘匿データを含めるべきではない。鍵を含める場合は鍵データを暗号化・難読化して容易に鍵データが取得されないように対策する必要がある。「アプリディレクトリ内」の root 権限を持つアプリやユーザーからも鍵を保護したい場合の対策も参照すること。

プロセスメモリ内の鍵の扱い

Android の持つ暗号技術を使用する場合、上図におけるアプリプロセス以外の場所で暗号化もしくは難読化されていた鍵データは暗号処理の手前で必ず復号 (パスワードベース鍵の場合は生成) が必要なため、プロセスメモリ内に鍵データが生のまま存在することになる。一方、アプリプロセスのメモリは通常別のアプリから読み書きすることはできないので、資産分類が本ガイドの対象範囲であれば、特別な対策を講じる必要性は少なく安全と言える。もし、アプリの扱う資産レベルや用途からプロセスメモリ内であっても鍵データがそのまま出現してはならない場合は、鍵データ・暗号ロジックの難読化など対策が必要であるが、一般に Java 層での実現は難しく、JNI 層での難読化ツールを使用することになる。これらの対応は本ガイドの範囲を超えるため、セキュア設計・実装の専門家と相談して対応すること。

5.6.3.4 Google Play 開発者サービスによる Security Provider の脆弱性対策

Google Play 開発者サービス（バージョン 5.0 以降）では、Provider Installer という仕組みが提供されており、Security Provider の脆弱性対策に利用できる。

まず、Security Provider とは、Java Cryptography Architecture（JCA）に基づいて各種暗号関連アルゴリズムの実装を提供するものである。Android アプリで暗号技術を用いる際には、Cipher、Signature、Mac といったクラスを通じてこの Security Provider を利用できる。一般に、暗号技術関連の実装で脆弱性が明らかになったときは迅速な対応が求められる。なぜなら脆弱性が悪用された際に大きな被害に繋がる可能性があるためである。Security Provider も暗号技術が関係するため、脆弱性への修正がいち早く反映されることが望ましい。

Security Provider の修正を反映する方法としては端末のアップデートが一般的である。端末のアップデートによる修正の反映は、メーカーがアップデートを用意し、ユーザーが用意されたアップデートを端末に適用するといったプロセスを経て行われる。そのため、アプリから、修正を含む最新の状態の Security Provider を使えるかどうかは、メーカーとユーザーの対応に左右される実態がある。それに対して、Google Play 開発者サービスの Provider Installer を利用すると、自動更新される Security Provider を使用できるようになる。

Google Play 開発者サービスの Provider Installer では、アプリから Provider Installer を呼び出すことで、Google Play 開発者サービスが提供する Security Provider を使用できるようになる。Google Play 開発者サービスは、Google Play ストアを通じて自動的にアップデートされるので、Provider Installer で提供される Security Provider も、メーカーとユーザーの対応状況に関わらず、最新の状態に更新されている。

以下に、Provider Installer を呼び出すサンプルコードを示す。

Provider Installer を呼び出す

```
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);
```

(次のページに続く)

(前のページからの続き)

```
        ProviderInstaller.installIfNeededAsync(this, this);
        setContentView(R.layout.activity_main);
    }

    @Override
    public void onProviderInstalled() {
        // Security Provider が最新、またはインストールが済んだときに呼ばれる
    }

    @Override
    public void onProviderInstallFailed(int errorCode, Intent recoveryIntent) {
        GoogleApiAvailability.getInstance().showErrorNotification(this, errorCode);

    }
}
```

5.6.3.5 Conscrypt モジュールについて

Conscrypt モジュールは、メーカーが用意する OTA アップデートに依存することなく、暗号技術関連の実装で明らかになった脆弱性への修正に利用できる APEX ファイルである。

Android 9 には、Conscrypt 用の Android 固有の公開 API は含まれていないが、Android 10 では、少数の公開 API メソッドが android.net.ssl に追加され、javax.net.ssl の下のクラスが非公開の Conscrypt 機能にアクセスできるようになっている。

Conscrypt モジュールは Google が OpenSSL から fork したネイティブ ライブラリ BoringSSL を使用しており、多くの Google プロダクトで暗号と TLS に使用されている。

もともと、以下のように特定のプロバイダを明示的にリクエストすることは非推奨であったが、特に BouncyCastle プロバイダで実装された暗号化アルゴリズムについては、Android 12 からは削除されている[*35]。

```
Cipher.getInstance("AES/CBC/PKCS7PADDING", "BC");
// OR
Cipher.getInstance("AES/CBC/PKCS7PADDING", Security.getProvider("BC"));
```

この変更に影響を受けるアプリは以下のようなものである。

1. KeyGenerator で無効な鍵サイズを使用しているアプリ
2. 初期化ベクトルのバイトサイズに 12 バイト以外のサイズを使用して Galois/Counter Mode（GCM）暗号を初期化しているアプリ

1 について、Conscrypt でサポートしている鍵サイズは 128 ビット、192 ビット、256 ビットとなっており、それ以外の鍵サイズを指定した場合、KeyGenerator.init() メソッド実行時に例外が発生する。このような場合、サポートされた適切な鍵サイズに修正する必要がある。

[*35] 執筆時点ではビルド時に警告が出ることもなく Android 12 エミュレータ上でも問題なく実行できることを確認しているが非推奨であることには違いはない

keygen.init(512, random); // 例外発生. Caused by: java.security.InvalidParameterException: Key size must be either 128, 192, or 256 bits

2 について、GCM 暗号を使用する場合、初期化ベクトルのバイトサイズは 12 バイトである必要がある。それ以外のサイズ、たとえば 16 バイトを指定していた場合、Cipher.init() メソッド実行時に例外が発生する。このような場合、サポートされた適切な初期化ベクトルのバイトサイズに修正する必要がある。

```
SecureRandom random = SecureRandom.getInstance("SHA1PRNG");
byte[] INITIALV = new byte[16]; // 初期化ベクトルのバイトサイズを 16 に指定
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暗号化アルゴリズム
cipher.init(Cipher.ENCRYPT_MODE, skey, ivp); // 例外発生. java.security.
↪InvalidAlgorithmParameterException: Expected IV length of 12 but was 16
```

5.6.3.6 バックアップデータの漏洩対策

攻撃者がバックアップファイルへのアクセス権を得ると、アプリでバックアップされた暗号化されていないデータが取得される可能性がある。アプリのバックアップ機能が不要と考えられる場合、バックアップの実行を防ぐためにマニフェストファイルに以下の設定を推奨する。

```
<application android:name="com.example.foo" android:allowBackup="false">
...
</application>
```

Android バージョン 12 以降では、allowBackup を false に設定すると、クラウドと ADB のバックアップは制限されるが、デバイス間の移行は許可される仕様となっている。

アプリの仕様としてバックアップが必要な場合、バックアップするデータの種類を以下のように指定する（Android 12 以降）。

```
<application android:name="com.example.foo"
  android:dataExtractionRules="backup_rules.xml">
 --
</application>
```

```
<application android:name="com.example.foo"
  android:fullBackupContent="@xml/backup_rules">
 --
</application>
```

詳細については以下の Android Developer ガイドを参照。[*36]

[*36] https://developer.android.com/privacy-and-security/risks/backup-leaks?hl=ja

5.6.3.7 アプリ内ハードコーディングのセキュリティリスク

開発者は暗号を使用してデータの機密性と完全性を保護することができるが、キーストレージを適切に活用せずにアプリのコードやアセットファイルにハードコードされているケースが多い。これにはセキュリティ上の重大な問題がある。

ハードコードされたクレデンシャルや暗号シークレットは、リバースエンジニアリングツールを使うことで簡単に取得されてしまう。これにより、攻撃者が機密データにアクセスすることが可能となり、セキュリティ上の重大なリスクが発生する。

そのため、システムレベルの認証情報が必要な場合は KeyChain API を使用し、各アプリ専用の認証情報は Android KeyStore プロバイダを利用して保存することが推奨される。特に、Android 16 以降では Secure Element 搭載端末において StrongBox 対応が必須となるケースが増えており、今後クレデンシャルを KeyChain や KeyStore に保存することが一般的な実践となることが予想される。これらの機能を活用することで、ハードコーディングによるリスクを効果的に低減できる。詳細は Android Developer ガイドを参照。[*37]

実装方法

1. Android KeyStore プロバイダを使用してアプリ固有の認証情報を保存

Android KeyStore を使用して対称鍵 (AES 鍵) を生成し保存する。

```
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)
```

(次のページに続く)

[*37] https://developer.android.com/privacy-and-security/risks/hardcoded-cryptographic-secrets

(前のページからの続き)

```
    aesKeyGenerator.generateKey()
}

@Throws(
    KeyStoreException::class,
    UnrecoverableEntryException::class,
    NoSuchAlgorithmException::class,
    CertificateException::class,
    IOException::class,
    NoSuchPaddingException::class,
    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. KeyChain API を使用してシステムレベルの認証情報を管理

KeyChain API を使用してシステム全体で共有される証明書と秘密鍵を管理する

```
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␣
```

(次のページに続く)

(前のページからの続き)

```
↪operations
                } catch (e: KeyChainException) {
                    e.printStackTrace()
                } catch (e: InterruptedException) {
                    e.printStackTrace()
                }
            }
        }
    }, null, null, null, -1, null)
}
```

3. クレデンシャルの取得と暗号化、証明書の選択

リモートサーバから API キーを取得し、暗号化した後、システムレベルの証明書を選択して使用する

```
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(", ")}")

    // Use KeyChain to select and use a system-level certificate
    selectCertificate()
} catch (e: Exception) {
    e.printStackTrace()
}
```

StrongBox 搭載端末の普及が今後も進むことで、KeyStore および KeyChain API との相乗効果が期待される。StrongBox はハードウェアベースのセキュリティモジュールとして、これらの API が管理する暗号鍵や認証情報の保護を強化し、全体的なセキュリティレベルを向上させる。StrongBox 未搭載の端末においても、KeyStore および KeyChain API を活用することで、Trusted Execution Environment（TEE）などのソフトウェアベースのセキュリティ機構を通じて十分なセキュリティ向上が可能である。将来的に StrongBox 搭載端末への移行が進むことを見越し、現段階からこれらの API を積極的に導入することが推奨される。これにより、ハードコーディングによるセキュリティリスクを効果的に低減し、堅牢なアプリケーション開発を継続的に実現できる。

5.6.3.8 Key sharing API によるアプリ間鍵共有の仕組みとリスク

Android 16 では、KeyStore で作成した鍵を、他のアプリと共有するための Key sharing API が追加された。KeyStoreManager クラスに、これらの API は設けられている。鍵の許可側アプリは、grantKeyAccess(String alias, int uid) で鍵へのアクセス許可、revokeKeyAccess(String alias, int uid) で鍵からアクセス許可のはく奪を行う。alias は作成した鍵のエイリアス、uid は鍵の使用を許可する被許可側アプリの uid である。鍵の被許可側アプリは、getGrantedKeyFromId(long id) または、getGrantedKeyPairFromId(long id) により、鍵を指定する。この id は、grantKeyAccess の返り値である。そのため、許可側アプリと被許可側アプリの間で、この id の受け渡しを行う必要がある。本来、Android KeyStore プロバイダを用いて作られた鍵は、生成したアプリ内でのみ使用できるということが特徴であり、Key sharing API はアプリのセキュリティレベルを低下させる恐れがある。そのため、Key sharing API の利用に際しては、以下の点に注意し、ユース

ケースを検討したうえでセキュリティ対策やリスクの受容を行う必要がある。

- 許可するアプリの信頼性の確認：許可するアプリの指定は、uid で行う。つまりユーザーの端末ごとに異なる値であることから、固定値での指定はできない。パッケージを指定して uid を確認する際には、インストールされているアプリが正当性を検証する必要がある。許可するアプリとの連携する場合は、4.1.2.10. 他社の特定アプリと連携する場合は利用先 *Activity* を確認する（必須） 等を参考に確認方法を実装する必要がある。
- 許可したアプリの管理：鍵の利用を許可した対象を取得するような API は提供されていない。そのため、許可したアプリを管理する必要がある場合は、独自にリストを作成し、保有しておく必要がある。
- 許可の期限：鍵の許可をいつまで継続させるかも検討する必要がある (自動的に許可を取り消す機能はない)。鍵を許可した後に、許可したアプリの更新時にマルウェアが紛れ込む、または脆弱性が発見され利用されることで、本来想定していない形で鍵へのアクセスが行われる可能性がある。そのため、ユースケースにもよるが、必要に応じて一定期間経過後に許可を取り消すようなロジックを組み込むことも検討されたい。

5.6.3.9 （コラム） Google Play アプリ署名による鍵管理

Android の Play アプリ署名（Google Play App Signing）は、Google Play ストアにアプリを公開する際に重要なセキュリティ機能である。アプリ署名は、開発者がアプリの APK ファイルにデジタル署名を付与するプロセスであり、このデジタル署名によってアプリが改ざんされていないことを証明できる。ユーザーがアプリをインストールする際、Android システムは署名を検証し、信頼できるアプリかどうかを判断する。

Play アプリ署名（Google Play App Signing）は、開発者がアプリの署名を Google に委任できるサービスである。このサービスを利用すると、Google がアプリのキー管理と署名を担い、より高いレベルのセキュリティを実現できる。開発者は最初にアップロードする APK にのみデジタル署名を行い、その後のバージョン管理や署名は Google が実施する。

署名の手順としては、まず開発者が自身のマシンでキーストアを生成し、その中の秘密鍵で APK ファイルにデジタル署名を行う。この作業は Android Studio やコマンドラインツールを使って実行できる。署名済み APK ファイルを Google Play コンソールにアップロードすることで、アプリの公開準備が整う。Google Play 署名サービスを利用する場合、最初のアップロードのみ開発者が署名し、その後のバージョンは Google が署名を代行する。

Google Play 署名サービスを利用することで、開発者はキー管理のリスクを軽減できる。Google は高度なセキュリティ対策を講じており、キーの安全性を確保しているため、不正アクセスや改ざんからアプリを保護できる。

また、Google Play 署名サービスで署名されたアプリは、ユーザーから信頼されやすくなる。これにより、ダウンロード数の増加やポジティブなレビュー獲得が期待できる。さらに、署名済みアプリは Google Play プロテクトによるスキャン対象となり、セキュリティが強化される。

Android の Play アプリ署名は、アプリのセキュリティと信頼性を確保するために不可欠な機能である。Google Play 署名サービスを活用することで、開発者はキー管理の負担を軽減し、ユーザーに信頼性の高いアプリを提供できる。この仕組みを理解し、適切に活用することが、安全なアプリ開発につながる。

なお、署名鍵の管理が不十分な場合、実際に流出や悪用が発生した事例も報告されている。具体的な事例については以下を参照。

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 アプリごとの Android Keystore（鍵）上限

Android 17 では、プラットフォームの安定性とセキュリティ向上を目的として、アプリごとに使用できる Android Keystore 内の鍵数に上限が導入された。具体的には、システムアプリ以外のアプリにつき最大で 50,000 個までの鍵（エントリ）しか作成できないという制限である。

この上限は、Android が提供する AndroidKeyStore システムに格納される鍵のみが対象となる。KeyGenParameterSpec などを用いて生成され、AndroidKeyStore プロバイダに保存された鍵がカウント対象であり、AES 鍵や RSA 鍵ペア、EC 鍵など、いずれも alias 単位で 1 つの鍵として数えられる。一方で、アプリの内部ストレージに保存される BKS や JKS 形式のファイル型キーストアや、独自の暗号ライブラリで管理する鍵などは、この上限には含まれない。あくまで OS が管理する AndroidKeyStore 領域のリソース制限という位置づけである。

この変更の背景には、アプリが意図せず大量の鍵を生成し続けることで、デバイス全体のリソースを圧迫する問題を防ぐ狙いがある。Android 17 ではメモリやリソース管理の強化が複数導入されており、その一環として Keystore の使用量にも制限が設けられている。上限を超える鍵を作成しようとすると、KeyStoreException が発生して作成に失敗する。

実務上、多くの一般的なアプリでは鍵の数は数個から数十個程度に収まるため、この上限に到達するケースはほとんどない。しかし、設計によっては注意が必要である。たとえば、ユーザーごとに個別の鍵を生成する設計や、ログインや処理ごとに新しい alias を作る実装の場合、ユーザー数や処理回数に比例して鍵が増加し、最終的に上限に達するリスクがある。

そのため、Android 17 以降では次のような設計が推奨される。まず、Keystore にはアプリ全体で共有する少数のマスター鍵を保存し、それ以外のデータ暗号化にはアプリ内部で生成した一時鍵やデータ鍵を使用する。そして、それらのデータ鍵をマスター鍵で暗号化（ラップ）することで、安全性とスケーラビリティを両立する。この方式であれば、Keystore 内の鍵数はほぼ一定に保たれ、上限に達する心配がない。

通常の実装では問題になりにくいものの、鍵の使い方次第では上限に到達する可能性があるため、alias の再利用や鍵管理の設計を見直すことが重要である。特に、キーを都度生成するような実装になっていないかを確認することが、今後の Android 対応において重要なポイントとなる。

5.7 生体認証機能を利用する

生体認証の分野では、顔、声紋をはじめ、様々な方式が研究・開発されている。中でも、指紋認証は本人を特定する方法として古くから存在しており、署名（拇印）や犯罪捜査の目的で利用されてきた。コンピューターの世界においても様々な分野で応用が進み、近年ではスマートフォンの所有者識別（主に画面ロック解除）に採用されるなど、（入力の容易さなど）利便性の高い機能としての認識が浸透しつつある。

そのような流れを受けて、Android 6.0(API Level 23) では端末に指紋認証のフレームワークが組み込まれ、アプリから指紋認証（本人確認）機能 (FingerprintManager) が利用できるようになった。さらに Android 9.0(API Level 28) では、指紋認証だけでなく、顔や虹彩認証などのそのほかの生体認証機能を統合的にサポートする BiometricPrompt API が導入された。また、これまでアプリが独自で用意することが必要であった認証用の UI は不要になり、標準的な認証用ダイアログが自動的に使用されるようになっている。これに合わせて従来の指紋認証機能 (FingerprintManager) は非推奨とされた。以下では、BiometricPrompt によって生体認証を行う際のセキュリティ上の注意事項を記す。

5.7.1 サンプルコード

生体認証機能には、利用者の認証情報に紐づけた鍵を使用する場合と、単純に利用者の認証のみを行う場合の大きく二つのユースケースがある。これらの生体認証の用途に応じて、 図 5.7.1 に従いサンプルコードを選択すること。

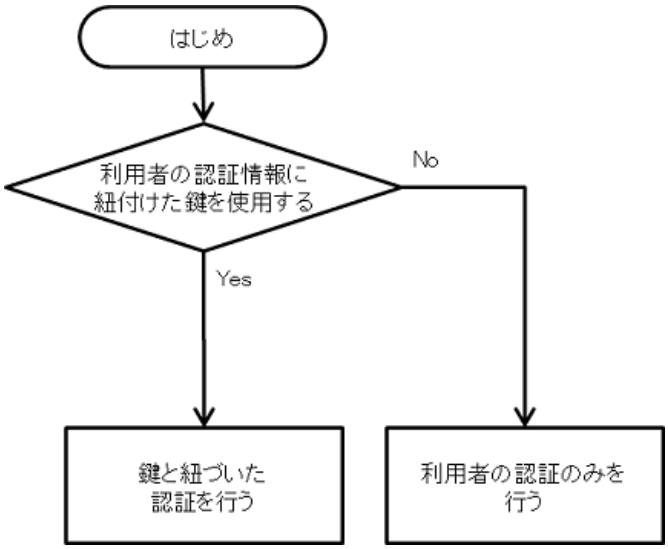

図 5.7.1　生体認証を利用するサンプルコード選択フローチャート

Android 9.0(API Level 28) がリリースされた時点ではまだ BiometricPrompt のサポートライブラリの提供が無く、したがって利用できるのは Android 9.0 搭載端末に限定されていた。しかし現在では サポートライブラリ [androidx.biometric](https://developer.android.com/reference/androidx/biometric/package-summary) が提供されており、これによって Android 6.0 以降のより広範な機種で BiometricPrompt を利用することができるようになっている。以下に示すサンプルコードではサポートライブラリとして提供されている BiometricPrompt を使用することとする。

5.7.1.1 鍵と紐づいた認証を行う

以下に利用者の認証情報に紐づけた鍵を使う場合に生体認証機能をアプリから利用するためのサンプルコードを示す。

ポイント:

1. USE_FINGERPRINT(Android 6.0〜Android 8.1) または USE_BIOMETRIC(Android 9.0〜) パーミッションを利用宣言する[*38]
2. ”AndroidKeyStore” Provider からインスタンスを取得する
3. 鍵を生成するためには生体情報の登録が必要である旨をユーザーに伝える
4. 鍵生成 (登録) 時、暗号アルゴリズムは脆弱でないもの（基準を満たすもの）を使用する
5. 鍵生成 (登録) 時、ユーザー (生体) 認証の要求を有効にする（認証の有効期限は設定しない）
6. 鍵を作る時点と鍵を使う時点で生体情報の登録状況が変わることを前提に設計を行う

[*38] サンプルコードで使用しているサポートライブラリ androidx.biometric の BiometricPrompt は Android 6.0(API Level 23)〜Android 8.1(API Level 27) デバイスでは FingerPrintManager の機能を使用して生体 (指紋) 認証を行うため、対応する USE_FINGERPRINT パーミッションの利用宣言が必要である。対して Android 9.0(API Level 28) 以降の端末では android.hardware.biometrics の BiometricPrompt の機能を使用しており、こちらは USE_BIOMETRIC パーミッションの利用宣言が必要となる (実際にはサポートライブラリパッケージ androidx.biometric のマニフェストファイルで、これらのパーミッションの利用宣言がされており、これを使用するアプリ側のマニフェストファイルで利用宣言をしなくとも問題なく動作する)。

7. 暗号化するデータは、生体認証以外の手段で復元（代替）可能なものに限る

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) {
```

(次のページに続く)

(前のページからの続き)

```
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        if (!isBiometricEnabled(this)) {
            // ★ポイント 3★ 鍵を生成するためには生体情報の登録が必要である旨をユーザーに伝える
            new AlertDialog.Builder(this)
                .setTitle(R.string.app_name)
                .setMessage("生体情報が登録されていません。 \n" +
                            "設定メニューの「セキュリティ」で生体情報を登録してください。\n" +
                            "生体情報を登録することにより、簡単に認証することができます。")
                .setPositiveButton("OK", null)
                .show();
            return;
        }

        // 指紋認証の結果を受け取るコールバック
        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 {
                    // ★ポイント 7★ 暗号化するデータは、指紋認証以外の手段で復元（代替）可能なものに限
る
                    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);
            }
        };
```

(次のページに続く)

(前のページからの続き)

```
        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) {
                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);
```

(次のページに続く)

(前のページからの続き)

```
        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
 *
 * 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.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;
```

(次のページに続く)

(前のページからの続き)

```
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 BiometricPrompt mBiometricPrompt;
    private BiometricPrompt.PromptInfo mPromptInfo;
    private CancellationSignal mCancellationSignal;
    private KeyStore mKeyStore;
    private KeyGenerator mKeyGenerator;
    private Cipher mCipher;


    public BiometricAuthentication(FragmentActivity context, final BiometricPrompt.
↪AuthenticationCallback callback) {
        // 指紋認証の結果を受け取るコールバック
        BiometricPrompt.AuthenticationCallback hook =
                new 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(BiometricPrompt.AuthenticationResult␣
↪result) {
                android.util.Log.e(TAG, "onAuthenticationSuccess");
                if (callback != null) callback.onAuthenticationSucceeded(result);
                reset();
            }

            @Override
```

(次のページに続く)

(前のページからの続き)

```
            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 BiometricPrompt(context, mExecutor, hook);
        final BiometricPrompt.PromptInfo.Builder builder =
            new BiometricPrompt.PromptInfo.Builder()
                .setTitle("Please Authenticate")
                .setNegativeButtonText("Cancel");
        mPromptInfo = builder.build();
        reset();
    }

    public boolean startAuthentication() {
        if (!generateAndStoreKey())
            return false;

        if (!initializeCipherObject())
            return false;

        BiometricPrompt.CryptoObject cryptoObject =
                new BiometricPrompt.CryptoObject(mCipher);

        // 指紋認証を実行
        android.util.Log.e(TAG, "Starting authentication");
        mBiometricPrompt.authenticate(mPromptInfo, cryptoObject);
        return true;
    }

    private void reset() {
        try {
            // ★ポイント 2 ★ "AndroidKeyStore" Provider からインスタンスを取得する
            mKeyStore = KeyStore.getInstance(PROVIDER_NAME);
            mKeyGenerator =
                    KeyGenerator.getInstance(KeyProperties.KEY_ALGORITHM_AES, PROVIDER_
```

(次のページに続く)

(前のページからの続き)

```
↪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);
        }
    }

    private boolean generateAndStoreKey() {
        try {
            mKeyStore.load(null);
            if (mKeyStore.containsAlias(KEY_NAME))
                mKeyStore.deleteEntry(KEY_NAME);
            mKeyGenerator.init(
                 // ★ポイント 4★ 鍵生成（登録）時、暗号アルゴリズムは脆弱でないもの（基準を満たすもの）
を使用する
                 new KeyGenParameterSpec.Builder(KEY_NAME, KeyProperties.PURPOSE_ENCRYPT)
                 .setBlockModes(KeyProperties.BLOCK_MODE_CBC)
                 .setEncryptionPaddings(KeyProperties.ENCRYPTION_PADDING_PKCS7)
                 // ★ポイント 5★ 鍵生成（登録）時、ユーザー（指紋）認証の要求を有効にする（認証の有効期
限は設定しない）
                 .setUserAuthenticationRequired(true)
                 .build());
            // 鍵を生成し、Keystore(AndroidKeyStore) に保存する
            mKeyGenerator.generateKey();
            return true;
        } catch (IllegalStateException e) {
            android.util.Log.e(TAG, e.getMessage());
            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);
            return false;
        }
    }

    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);
```

(次のページに続く)

(前のページからの続き)

```
            KeyInfo info = (KeyInfo) factory.getKeySpec(key, KeyInfo.class);

            mCipher.init(Cipher.ENCRYPT_MODE, key);
            return true;
        } catch (KeyPermanentlyInvalidatedException e) {
            // ★ポイント 6★ 鍵を作る時点と鍵を使う時点で指紋の登録状況が変わることを前提に設計を行う
            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"
          >

  <!-- ★ポイント 1★ USE_FINGERPRINT(Android 6.0〜Android 8.1) または USE_BIOMETRIC(Android 9.
↪0〜) パーミッションを利用宣言する -->
  <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 利用者の認証のみ行う

以下に、利用者による認証のみ行う場合に生体認証機能を利用するためのサンプルコードを示す。この場合、特にセキュリティ上注意すべき点はないが、参考のためサンプルコードを以下に掲載する。

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);
```

(次のページに続く)

(前のページからの続き)

```
        button_biometric_auth.setOnClickListener(new View.OnClickListener () {
            @Override
            public void onClick(View v) {
                if (!mBiometricAuthentication.isAuthenticating()) {
                    authenticateByBiometric();
                }
            }
        });
    }

    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;
    }
```

(次のページに続く)

(前のページからの続き)

```
    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

```
/*
 * 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;
```

(次のページに続く)

(前のページからの続き)

```
    private Context mContext;

    // ★ ユーザーが「キャンセル」ボタンをタッチした際の処理
    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);
        // 認証プロンプトではキャンセル用のボタンも表示される
        // キャンセル処理は setNegativeButton の第 3 引数に与える
        // DialogInterface.OnClickListener で実施する
        mBiometricPrompt = builder
            .setTitle("Please Authenticate")
            .setNegativeButton("Cancel",context.getMainExecutor() ,cancelListener)
            .build();
        reset();
    }

    public boolean startAuthentication(
            final BiometricPrompt.AuthenticationCallback callback) {

        mCancellationSignal = new CancellationSignal();

        // 指紋認証の結果を受け取るコールバック
        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();
            }
```

(次のページに続く)

(前のページからの続き)

```
            @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();
                }
            }
        };

        // 指紋認証を実行
        android.util.Log.d(TAG, "Starting authentication");
        mBiometricPrompt.authenticate(mCancellationSignal,
                                      mContext.getMainExecutor(),
                                      hook);

        return true;
    }

    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"
          >

  <!-- ★ポイント 1 ★ USE_FINGERPRINT(Android 6.0〜Android 8.1) または USE_BIOMETRIC(Android 9.
↪0〜) パーミッションを利用宣言する -->
  <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 ルールブック

暗号鍵と紐づいた認証を行う際に生体機能を利用する場合には以下のルールを守ること。それ以外の用途で生体認証機能を利用する場合は、特にルールはない。

1. 鍵生成 (登録) 時、暗号アルゴリズムは脆弱でないもの（基準を満たすもの）を使用する （必須）
2. 暗号化するデータは、生体認証以外の手段で復元（代替）可能なものに限る （必須）
3. 鍵を生成するためには生体情報の登録が必要であり、ユーザーにその旨を伝える （推奨）

5.7.2.1 鍵生成 (登録) 時、暗号アルゴリズムは脆弱でないもの（基準を満たすもの）を使用する （必須）

「5.6. 暗号技術を利用する」で解説したパスワード鍵や公開鍵等と同様に、生体認証機能を用いて鍵を生成する場合においても、第三者による盗聴を防ぐため一定の基準を満たした脆弱でない暗号アルゴリズムを設定すること。また、暗号アルゴリズムだけではなく、暗号モードやパディングについても脆弱でない安全な方式を選択すること。

暗号アルゴリズムの選択については「5.6.2.2. 脆弱でないアルゴリズム（基準を満たすもの）を使用する （必須）」を参照すること。

5.7.2.2 暗号化するデータは、生体認証以外の手段で復元（代替）可能なものに限る （必須）

アプリ内データの暗号化に際して生体認証機能を利用する場合は、そのデータが生体認証機能以外の手段でも復元（代替）できるようにアプリを設計すること。一般に、生体情報には秘匿や変更の困難性、誤認識といった問題が付随しているため、認証を生体情報のみに頼ることは避けるべきである。

例えば、生体認証機能を利用して生成した鍵を用いてアプリ内のデータを暗号化した後、端末に登録されていた生体情報がユーザーによって削除されると、データの暗号化に用いていた鍵が使用できなくなり、データを復号することもできなくなる。データの復元が生体認証機能以外の手段でも行えなければ、可用性が損なわれる可能性が生じるのである。また、生体認証機能を利用して生成した鍵が使用できなくなる状況は生体情報の削除以外でも生じうる。例えば本来正しく使用できるべき鍵が指紋センサーの誤認識あるいはセンサーの不具合によって使用できなくなる可能性もある。

5.7.2.3 鍵を生成するためには生体情報の登録が必要であり、ユーザーにその旨を伝える （推奨）

生体認証機能を利用して鍵を生成するためには端末にユーザーの生体情報が登録されている必要がある。生体情報の登録をユーザーに促すために設定メニューへ誘導する際には、生体情報が重要な個人情報であることに留意し、ユーザーに対してアプリが生体情報をを利用する必要性や利便性について説明することが望ましい。

指紋認証の登録が必要である旨、ユーザーに伝える

```
if (!mFingerprintAuthentication.isFingerprintAuthAvailable()) {
    // ★ポイント★ 鍵を生成するためには生体情報の登録が必要であり、ユーザーにその旨を伝える
    new AlertDialog.Builder(this)
        .setTitle(R.string.app_name)
        .setMessage("生体情報が登録されていません。\n" +
                    "設定メニューの「セキュリティ」で指紋などの生体を登録してください。\n" +
                    "生体情報を登録することにより、簡単に認証することができます。")
        .setPositiveButton("OK", null)
        .show();
    return false;
}
```

5.7.3 アドバンスト

5.7.3.1 Android アプリにおける生体認証機能利用の前提条件

アプリで生体認証機能を利用するためには以下の 2 点が必要である。

- 端末にユーザーの生体情報が登録されていること
- 登録された生体情報に（アプリ固有の）鍵が紐づいていること

ユーザーの生体情報登録

ユーザーの生体情報は設定メニューの「セキュリティ」からしか登録することができず、一般のアプリが生体情報の登録処理を行うことはできない。そのため、アプリが生体認証機能を利用する時点で端末に生体情報が登録されていなければ、ユーザーを設定メニューに誘導し生体情報の登録を促す必要がある。その際、アプリが生体情報を利用する必要性や利便性についての説明がユーザーに対して行われることが望ましい。

なお、生体認証が可能となる前提として、端末に予備の画面ロック方式が設定されていることが必要である。端末に生体情報が登録されている状態で画面ロックを無効にすると、登録済みの生体情報も削除される。

鍵の生成・登録

端末に登録された生体情報と鍵を紐づけるためには、”AndroidKeyStore” Provider が提供する KeyGenerator や KeyStore インスタンス等を使用する際に、ユーザー（生体）認証の要求を有効にして、新しい鍵の生成と登録、あるいは既存の鍵の登録を行う。

生体情報と紐づいた鍵を生成する場合は、KeyGenerator を生成する際のパラメータとして、ユーザー認証の要求を有効にするよう設定する。

生体情報と紐づいた鍵の生成と登録

```
try {
    // "AndroidKeyStore" Provider から KeyGenerator インスタンスを取得する
    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)
        .setUserAuthenticationRequired(true)  // ユーザー（生体）認証の要求を有効にする
        .build());
    keyGenerator.generateKey();
} catch (IllegalStateException e) {
    // 端末に生体情報が登録されていない
  throw new RuntimeException("No fingerprint registered", e);
} catch (NoSuchAlgorithmException | InvalidAlgorithmParameterException
            | CertificateException | KeyStoreException | IOException e) {
    // 鍵の生成に失敗
    throw new RuntimeException("Failed to generate a key", e);
}
```

既存の鍵に生体情報を紐づける場合は、ユーザー認証の要求を有効にする設定を加えた KeyStore エントリーに鍵を登録する。

既存の鍵に生体情報を紐づける

```
SecretKey key = existingKey;  // 既存の鍵

KeyStore keyStore = KeyStore.getInstance("AndroidKeyStore");
keyStore.load(null);
keyStore.setEntry(
    "alias_for_the_key",
    new KeyStore.SecretKeyEntry(key),
    new KeyProtection.Builder(KeyProperties.PURPOSE_ENCRYPT)
            .setUserAuthenticationRequired(true)  // ユーザー（生体）認証の要求を有効にする
            .build());
```

5.7.3.2 Android 11 における生体認証の変更点

Android 11 では生体認証において以下の 3 点が変更となっている。

- BiometricManager.Authenticators インターフェースの導入
- BiometricPrompt におけるデータアクセスの強化
- サポート終了のメソッド

BiometricManager.Authenticators インターフェース

BiometricManager クラスおよび BiometricPrompt クラスでサポートされる認証のタイプは BiometricManager.Authenticators インターフェースで以下のように定義されている。

表 5.7.1　Authenticators インターフェース

認証タイプ	解説
BIOMETRIC_STRONG	強度レベルが Strong となるハードウェア要素を使用する認証
BIOMETRIC_WEAK	強度レベルが Weak となるハードウェア要素を使用する認証
DEVICE_CREDENTIAL	画面ロックの認証情報（ユーザーの PIN、パターン、パスワード）を使用する認証

アプリがどの認証タイプを受け入れるかは、setAllowedAuthenticators() メソッドに上記の認証タイプを引数として渡し定義する。渡せる認証タイプは単一でも複数でも可能となっており、たとえば強度レベルが Strong のハードウェア要素と画面ロックの認証情報を受け入れるよう定義するには、BIOMETRIC_STRONG | DEVICE_CREDENTIAL を引数として渡す。

アプリが必要とする認証要素が使用可能かどうかを確認するには canAuthenticate() メソッドにより行う。このとき PIN、パターン、パスワードをユーザーが作成していない場合、ACTION_BIOMETRIC_ENROLL インテントのアクションを呼び出す。このインテントは、アプリが受け入れる認証システムの認証情報を登録するようにユーザーに求める。

ユーザー認証の後、getAuthenticationType() メソッドを実行することで、認証されたユーザーがデバイスの認証情報と生体認証情報のどちらを使用したかを確認することができる。

BiometricPrompt におけるデータアクセスの強化

Android 11 では、BiometricPrompt におけるデータアクセスが強化され、支払情報や医療情報など、秘匿性が高く鍵で保護されたユーザーデータにアプリがアクセスするたびに、その都度、生体認証情報もしくはデバイスの認証情報を求める必要がある。

既存の鍵と BiometricPrompt オブジェクトを紐づける際のコードを以下に示す。

```
KeyGenParameterSpec authPerOpKeyGenParameterSpec =
        new KeyGenParameterSpec.Builder(KEY_NAME, KeyProperties.PURPOSE_ENCRYPT)
    // 生体認証情報もしくはデバイスの認証情報を受け入れる
    .setUserAuthenticationParameters(0, KeyProperties.AUTH_BIOMETRIC_STRONG |␣
↪KeyProperties.AUTH_DEVICE_CREDENTIAL)
    .build();
```

サポート終了のメソッド

Android 11 では、以下のメソッドのサポートが終了し利用できない。

- setDeviceCredentialAllowed() メソッド
- setUserAuthenticationValidityDurationSeconds() メソッド
- オーバーロードで定義された引数のない canAuthenticate() メソッド

6 難しい問題

Android には OS の仕様や OS が提供する機能の仕様上、アプリの実装でセキュリティを担保するのが困難な問題が存在する。これらの機能は悪意を持った第三者に悪用されたり、ユーザーが注意せずに利用したりすることで、情報漏洩を始めセキュリティ上の問題に繋がってしまう危険性を常に抱えている。この章ではそのような機能に対して、開発者が取りうるリスク削減策などを提示しながら注意喚起が必要な話題を記事として取り上げる。

6.1 Clipboard から情報漏洩する危険性

コピー＆ペーストはユーザーが普段から何気なく使っている機能であろう。例えば、この機能を使って、メールや Web ページで気になった情報や忘れたら困る情報をメモ帳に残しておいたり、設定したパスワードを忘れないようにメモ帳に保存しておき、必要な時にコピー＆ペーストして使うというユーザーは少なからず存在する。これらは一見何気ない行為であるが、実はユーザーの扱う情報が盗まれるという危険が潜んでいる。

これには Android のコピー＆ペーストの仕組みが関係している。ユーザーやアプリによってコピーされた情報は、一旦 Clipboard と呼ばれるバッファに格納される。ユーザーやアプリによってペーストされたときに、この Clipboard の内容が各アプリに再配布されるわけである。この Clipboard に情報漏洩に結び付く危険性がある。Android 端末の仕様では、Clipboard の実体は端末に 1 つであり、ClipboardManager を利用することで、どのアプリからでも常時 Clipboard の中身が取得できるようになっているからである。このことは、ユーザーがコピー・カットした情報は全て悪意あるアプリに対して筒抜けになることを意味している。

よって、アプリ開発者は、この Android の仕様を考慮しながら情報漏洩の可能性を最小限に抑える対策を講じなくてはならない。

6.1.1 サンプルコード

Clipboard から情報漏洩する可能性を抑える対策には、大きく分けて次の 2 つが考えられる。

(1) 他アプリから自アプリへコピーする際の対策

(2) 自アプリから他アプリへコピーする際の対策

最初に、1. について説明する。ここでは、ユーザーがメモ帳や Web ブラウザ、メーラーアプリなど他アプリから文字列をコピーし、それを自アプリの EditText に貼り付けるシナリオを想定している。結論だけを言ってしまうと、このシナリオでコピー・カットによってセンシティブな情報が漏洩してしまうことを防ぐ根本的な対策は存在しない。第三者アプリのコピー機能を制御するような機能が Android にはないからだ。

よって、1. についてはセンシティブな情報をコピー・カットする危険性をユーザーに説明し、行為自体を減らしていく啓発活動を継続的に行っていくしか対策はない。

次に、2. を説明する。ここでは、自アプリが表示している情報がユーザーによってコピーされるシナリオを想定する。この場合、漏洩に対する確実な対策は、View(TextView, EditText など) からのコピー・カットを禁止にすることである。個人情報などセンシティブな情報が入力あるいは出力される View にコピー・カット機能がなければ、自アプリからの Clipboard を介した情報の漏洩もないからだ。

コピー・カットを禁止する方法はいくつか考えられるが、ここでは、実装が簡単でかつ効果のある方法として、View の長押し無効化の方法と文字列選択時のメニューからコピー・カットの項目を削除する方法を扱う。

対策要否は、図 6.1.1 の判定フローによって判定することができる。図 6.1.1 において、入力タイプ (Input Type) が Password 属性に固定されているとは、入力タイプ (Input Type) がアプリの実行時に常に下記のいずれかであることを指す。この場合は、デフォルトでコピー・カットが禁止されているので、特に対策する必要はない。

- 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

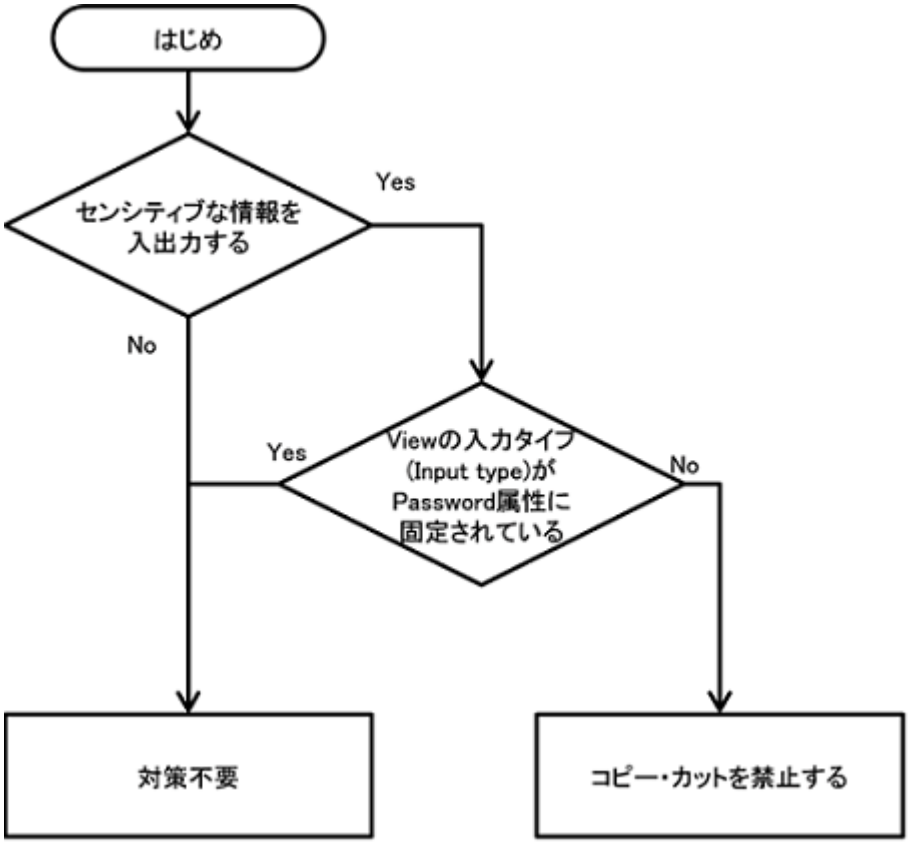

図 6.1.1　対策要否の判定フロー

以下で、それぞれの対策の詳細を説明し、サンプルコードを示す。

6.1.1.1 文字列選択時のメニューからコピー・カットを削除する

TextView.setCustomSelectionActionModeCallback() メソッドによって、文字列選択時のメニューをカスタマイズできる。これを用いて、文字列選択時のメニューからコピー・カットのアイテムを削除すれば、ユーザーが文字列をコピー・カットすることはできなくなる。

以下、EditText の文字列選択時のメニューからコピー・カットの項目を削除するサンプルコードを示す。

ポイント：

1. 文字列選択時のメニューから android.R.id.copy を削除する。
2. 文字列選択時のメニューから android.R.id.cut を削除する。

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);
```

(次のページに続く)

(前のページからの続き)

```
        setContentView(R.layout.uncopyable);

        copyableEdit = (EditText) findViewById(R.id.copyable_edit);
        uncopyableEdit = (EditText) findViewById(R.id.uncopyable_edit);
        // setCustomSelectionActionModeCallback メソッドにより、
        // 文字列選択時のメニューをカスタマイズすることができる。
        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) {
                // ★ポイント 1 ★ 文字列選択時のメニューから android.R.id.copy を削除する。
                MenuItem itemCopy = menu.findItem(android.R.id.copy);
                if (itemCopy != null) {
                    menu.removeItem(android.R.id.copy);
                }
                // ★ポイント 2 ★ 文字列選択時のメニューから android.R.id.cut 削除する。
                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) {
```

(次のページに続く)

(前のページからの続き)

```
        switch (item.getItemId()) {
        case android.R.id.home:
            NavUtils.navigateUpFromSameTask(this);
            return true;
        }
        return super.onOptionsItemSelected(item);
    }

}
```

6.1.1.2 View の長押し (Long Click) を無効にする

コピー・カットを禁止する方法は、View の長押し (Long Click) を無効にすることでも実現できる。View の長押し無効化はレイアウトの xml ファイルで指定することができる。

ポイント：

1. コピー･カットを禁止する View は android:longClickable を false にする。

```
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" />

    <!-- コピー・カットを禁止する EditText -->
    <!-- ★ポイント 1★ コピー・カットを禁止する View は android:longClickable を false にする。 -->
    <EditText
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:longClickable="false"
        android:hint="@string/unlongclickable_hint" />
</LinearLayout>
```

6.1.2 ルールブック

自アプリから他アプリへのセンシティブな情報のコピーが発生する可能性がある場合は、以下のルールを守ること。

1. *View* に表示されている文字列のコピー・カットを無効にする *(必須)*

6.1.2.1 View に表示されている文字列のコピー・カットを無効にする (必須)

アプリがセンシティブな情報を表示する View を持っている場合、それが EditText のようにコピー・カットが可能な View ならば、Clipboard を介してその情報が漏洩してしまう可能性がある。そのため、センシティブな情報を表示する View はコピー・カットを無効にしておかなければならない。

コピー・カットを無効にする方法には、文字列選択時のメニューからコピー・カットの項目を削除する方法と、View の長押しを無効化する方法がある。

「6.1.3.1. ルール適用の際の注意」も参照のこと。

6.1.3 アドバンスト

6.1.3.1 ルール適用の際の注意

TextView はデフォルトでは文字列選択不可であるため、通常は対策不要であるが、アプリの仕様によってはコピーを可能にする場合もある。TextView.setTextIsSelectable() メソッドを使うことで、文字列の選択可否とコピー可否を動的に設定することができる。TextView をコピー可能とする場合は、その TextView にセンシティブな情報が表示される可能性がないかよく検討し、その可能性があるのであれば、コピー可にすべきでない。

また、「6.1.1. サンプルコード」の判定フローにも記載されているように、パスワードの入力を想定した入力タイプ (InputType.TYPE_CLASS_TEXT|InputType.TYPE_TEXT_VARIATION_PASSWORD など) の EditText については、デフォルトで文字列のコピーが禁止されているため通常は対策不要である。しかし、アプリ独自に「パスワードを平文表示する」機能などを実装して入力タイプを動的に変更している場合は、平文表示の際にコピー・カットが有効になってしまう可能性があるため、同様の対策が必要である。

なお、ルールを適用する際には、ユーザビリティの面も考慮する必要があるだろう。例えば、ユーザーが自由にテキストを入力できる View の場合、センシティブな情報が入力される「可能性がゼロでない」からといってコピー・カットを無効にしてしまったら、ユーザーの使い勝手が悪くなるだろう。もちろん、重要度の高い情報を入出力する View やセンシティブな情報を単独で入力するような View にはルールを無条件で適用するべきであるが、それ以外の View を扱う場合は、次のことを考慮しながら対応を考えると良い。

- センシティブな情報の入力や表示を行う専用のコンポーネントを用意できないか
- 連携先 (ペースト先) アプリが分かっている場合は、他の方法で情報を送信できないか
- アプリでユーザーに入出力に関する注意喚起ができないか
- 本当にその View が必要か

Android OS の Clipboard と ClipboardManager の仕様にセキュリティに対する考慮がされていないことが情報漏洩の可能性を生む根本的な要因ではあるが、アプリ開発者は、ユーザー保護やユーザビリティ、提供する機能など様々な観点からこうした Clipboard の仕様に対して対応し、質の高いアプリを作成する必要がある。

6.1.3.2 Clipboard に格納されている情報の操作

続く内容は Android 9 以前の端末に適用される。Android 10 からはユーザーのプライバシー保護のため、Clipboard の取得に制限がかけられたことで、Clipboard にアクセスするには、アプリがデフォルトの IME であるか、フォーカスを持っている必要がある。サンプルプログラムの動作確認を行う場合、Android 9 以前の端末で行うこと。[*1]

[*1] https://developer.android.com/about/versions/10/privacy/changes

「6.1. *Clipboard* から情報漏洩する危険性」で述べたように、ClipboardManager を利用することでアプリから Clipboard に格納された情報を操作することができる。また、ClipboardManager の利用には特別な Permission を設定する必要が無いため、アプリはユーザーに知られることなく ClipboardManager を利用できる。

Clipboard に格納されている情報 (ClipData と呼ぶ) は、ClipboardManager.getPrimaryClip() メソッドによって取得できる。タイミングに関しても、OnPrimaryClipChangedListener を実装して ClipboardManager.addPrimaryClipChangedListener() メソッドで ClipboardManager に登録すれば、ユーザーの操作などにより発生するコピー・カットの度に Listener が呼び出されるので、タイミングを逃すことなく ClipData を取得することができる。ここで Listener の呼び出しは、どのアプリでコピー・カットが発生したかに関係なく行われる。

以下、端末内でコピー・カットが発生する度に ClipData を取得し、Toast で表示する Service のソースコードを示す。下記のような簡単なコードにより Clipboard に格納された情報が筒抜けになってしまうことを実感していただきたい。アプリを実装する際は、少なくとも下記のコードによってセンシティブな情報が取得されてしまうことのないように注意する必要がある。

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";
    private ClipboardManager mClipboardManager;
```

(次のページに続く)

(前のページからの続き)

```
    @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, "ClipboardService の取得に失敗しました。サービスを終了します。");
            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(),
                        "コピーあるいはカットされた文字列:\n"
                        + item.coerceToText(getApplicationContext()),
                        Toast.LENGTH_SHORT)
                    .show();
            }
        }
    };
}
```

次に、上記 ClipboardListeningService を利用する Activity のソースコードの例を示す。

ClipboardListeningActivity.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.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;
import android.widget.Toast;

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 が不正です");
        } else {
            ComponentName cn =
                startService(new Intent(ClipboardListeningActivity.this,
                                        ClipboardListeningService.class));
            if (cn == null) {
                Log.e(TAG, "サービスの起動に失敗しました");
```

(次のページに続く)

(前のページからの続き)

```
                Toast.makeText(this, "サービスの起動に失敗しました", Toast.LENGTH_SHORT).show();
            }
        }
    }

    public void onClickStopService(View view) {
        if (view.getId() != R.id.stop_service_button) {
            Log.w(TAG, "View ID が不正です");
        } else {
            stopService(new Intent(ClipboardListeningActivity.this,
                                   ClipboardListeningService.class));
        }
    }
}
```

ここまでは、Clipboard に格納された情報を取得する方法について述べたが、ClipboardManager.setPrimaryClip() メソッドによって、Clipboard に新しく情報を格納することも可能である。

ただし、setPrimaryClip() は Clipboard に格納されていた情報を上書きするので、ユーザーが予めコピー・カット操作により格納しておいた情報が失われる可能性がある点に注意が必要である。これらのメソッドを使用して独自のコピー機能あるいはカット機能を提供する場合は、必要に応じて、内容が改変される旨を警告するダイアログを表示するなど、Clipboard に格納されている内容がユーザーの意図しない内容に変更されることのないように設計・実装する必要がある。

また、パスワードやクレジットカード情報などの機密コンテンツを、setPrimaryClip() でクリップボードにコピーするアプリを Android 13 で動作させる場合は、setPrimaryClip() を呼び出す前に ClipDescription でフラグを設定する必要がある。これはユーザーのプライバシー保護の観点から Android 13 で導入された仕様であり、このフラグを設定することで、コンテンツのプレビューに機密コンテンツが表示されなくなる。このフラグの設定有無による違いを以下に示す。ボタン押下時にパスワード文字列を setPrimaryClip() でコピーするアプリの様子である。

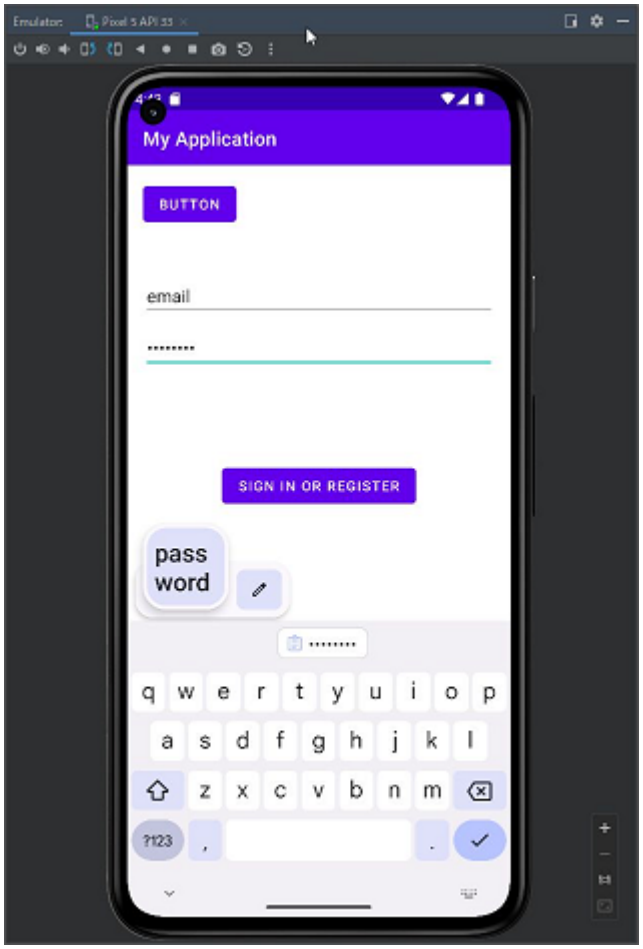

図 6.1.2　ClipDescription フラグ設定なし

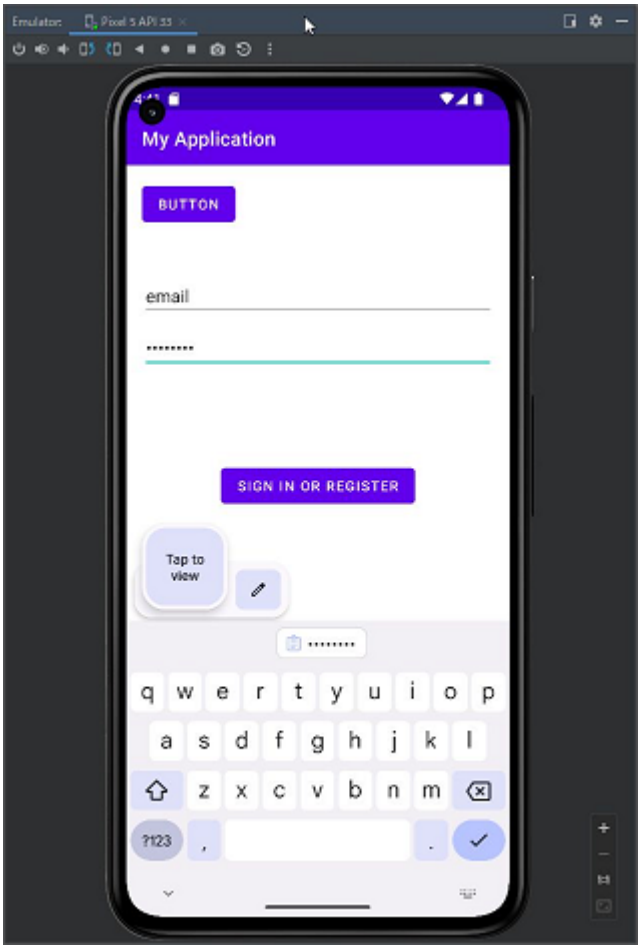

図 6.1.3　ClipDescription フラグ設定あり

ClipDescription フラグの設定方法は、SDK バージョンが API level 33 以上かどうかで違いがある。サンプルコードを以下の通り。実際に ClipDescription フラグを設定する場合は、コード中のコメントから適宜、使用している SDK バージョンに合わせること。

```
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以上
// extras.putBoolean("android.content.extra.IS_SENSITIVE", true); // lower API level
cd.getDescription().setExtras(extras);

cm.setPrimaryClip(cd);
```

6.1.3.3 Clipboard へのアクセス通知について

Android 12 から、 別のアプリが Clipboard に格納したデータを getPrimaryClip() で取得すると、ユーザーに Clipboard へのアクセスが行われたことを通知する Toast メッセージが表示される。通知メッセージは以下のテキストになる。

```
MYAPP にクリップボードから貼り付けました
```

また、getPrimaryClipDescription() を使用した場合は、データのコピーは行われず、Toast メッセージも表示されない。getPrimaryClipDescription() で取得できる情報は、テキストがスタイル設定されたものかどうかの真偽値や、URL などテキストの各種分類など、格納されたデータそのものの情報となる。

更新履歴

2012-06-01 初版

2012-11-01

下記の構成・内容を見直し拡充致しました

- 4.1. *Activity* を作る・利用する
- 4.2. *Broadcast* を受信する・送信する
- 4.3. *Content Provider* を作る・利用する
- 4.4. *Service* を作る・利用する
- 5.2. *Permission* と *Protection Level*

下記の新しい記事を追加致しました

- 2.5. サンプルコードの *Android Studio* への取り込み手順
- 3.1. *Android* アプリのセキュリティ
- 4.7. *Intent* を利用する
- 5.3. *Account Manager* に独自アカウントを追加する
- 6.1. *Clipboard* から情報漏洩する危険性

2013-04-01

下記の記事の内容を見直し書き直しました

- 5.3. *Account Manager* に独自アカウントを追加する

下記の新しい記事を追加致しました

- 4.8. *LogCat* にログ出力する
- 5.4. *HTTPS* で通信する
- 4.9. *WebView* を使う

2014-07-01

下記の新しい記事を追加致しました

- 5.5. プライバシー情報を扱う
- 5.6. 暗号技術を利用する

2015-06-01

下記の方針で本書全体の内容を見直し書き直しました

- 開発環境の変更 (Eclipse -> Android Studio)
- Android 最新版 Lollipop への対応
- 対応する API Level の見直し (8 以降 -> 15 以降)

2016-02-01

下記の新しい記事を追加致しました

- 4.10. *Notification* を使用する
- 5.7. 生体認証機能を利用する

下記の構成・内容を見直し拡充致しました

- 5.2. *Permission* と *Protection Level*

2016-09-01

下記の構成・内容を見直し拡充致しました

- 2.5. サンプルコードの *Android Studio* への取り込み手順
- 5.4. *HTTPS* で通信する
- 5.6. 暗号技術を利用する

2017-02-01

下記の新しい記事を追加致しました

- 5.4.3.7. *Network Security Configuration*

下記の構成・内容を見直し拡充致しました

- 4.1. *Activity* を作る・利用する
- 4.2. *Broadcast* を受信する・送信する
- 4.4. *Service* を作る・利用する
- 4.5. *SQLite* を使う
- 4.6. ファイルを扱う

下記の記事を削除致しました

- 4.8.3.4 BuildConfig.DEBUG は ADT 21 以降で使う

下記の方針で本書全体の内容を見直し書き直しました

- Android 4.0.3（API Level 15）未満に関する本文中の記述を削除または脚注へ移動

2018-02-01

下記の新しい記事を追加致しました

- 4.1.3.6. *Autofill* フレームワークについて
- 5.3.3.2. *Android 8.0*（*API Level 26*）以降で署名の一致しない *Authenticator* のアカウントを読めるケース
- 5.4.3.8. （コラム） セキュア接続の *TLS1.2/TLS1.3* への移行について
- 5.5.3.3. *Android ID* のバージョンによる違いについて

下記の構成・内容を見直し拡充致しました

- 4.2. *Broadcast* を受信する・送信する
- 5.2. *Permission* と *Protection Level*
- 5.3. *Account Manager* に独自アカウントを追加する
- 5.4. *HTTPS* で通信する
- 5.5. プライバシー情報を扱う

2018-09-01

下記の新しい記事を追加いたしました

- 4.9.3.3. *WebView* の *Safe Browsing* について
- 4.11. 共有メモリを使用する

下記の構成・内容を見直し拡充いたしました

- 2.5. サンプルコードの *Android Studio* への取り込み手順
- 4.1.3.6. *Autofill* フレームワークについて
- 4.5.3.6. *[参考]SQLite* データベースを暗号化する *(SQLCipher for Android)*
- 5.2.1.2. 独自定義の *Signature Permission* で自社アプリ連携する方法
- 5.4.1.2. *HTTPS* 通信する
- 5.4.3.7. *Network Security Configuration*
- 5.7. 生体認証機能を利用する
- 5.4.3.2. *Android OS* の証明書ストアにプライベート認証局のルート証明書をインストールする
- 5.4.3.8. （コラム） セキュア接続の *TLS1.2/TLS1.3* への移行について

2019-09-01

下記の新しい記事を追加いたしました

- 4.6.3.5. *Android 10* 以降の外部ストレージアクセス

- 5.5.3.4. *Android 10* での再設定不可能なデバイス *ID* の取得制限について

下記の構成・内容を見直し拡充いたしました

- 4.6.1.4. 外部記憶 *(*読み書き公開*)* ファイルを扱う
- 5.2.3.6. *Android 6.0* 以降の *Permission* モデルの仕様変更について
- 5.4.1.1. *HTTP* 通信する
- 5.4.1.2. *HTTPS* 通信する
- 5.4.3.8. （コラム） セキュア接続の *TLS1.2/TLS1.3* への移行について
- 5.5.1.2. *[*包括同意あり*]* アプリ・プライバシーポリシーを組み込んだアプリ
- 5.6.2.2. 脆弱でないアルゴリズム（基準を満たすもの）を使用する （必須）
- 5.6.3.2. 乱数の生成
- 5.7. 生体認証機能を利用する

2019-12-01

下記の構成・内容を見直し拡充いたしました

- 4.1.3.1. *exported* 設定と *intent-filter* 設定の組み合わせ *(Activity* の場合*)*
- 4.4.1.2. 公開 *Service* を作る・利用する
- 4.4.3.1. *exported* 設定と *intent-filter* 設定の組み合わせ *(Service* の場合*)*

2020-11-01

下記の新しい記事を追加いたしました

- 5.2.3.7. *Android 11* 以降での使用していないアプリの権限が自動リセットされる機能について
- 5.5.3.5. データアクセスの監査について
- 5.5.3.6. 位置情報アクセスについて
- 5.6.3.5. *Conscrypt* モジュールについて
- 5.7.3.2. *Android 11* における生体認証の変更点

下記の構成・内容を見直し拡充いたしました

- 3.1.1.2. *Android* スマートフォンにおける機能資産
- 5.2.3.6. *Android 6.0* 以降の *Permission* モデルの仕様変更について

2021-10-19

下記の新しい記事を追加いたしました

- 4.2.3.7. *ACTION_CLOSE_SYSTEM_DIALOGS* について
- 4.10.3.2. 特定の *Window* をパススルーするタッチについて
- 4.7.3.2. *PendingIntent* オブジェクトの可変性

- 5.5.3.8. *Android 12* における使用していないアプリの自動休止機能について
- 5.5.3.9. パッケージアクセスの仕様変更にともなう *API* の戻り値の変化
- 5.5.3.10. *Android 12* におけるマイクとカメラ
- 5.5.3.11. *WebView* における *SameSite Cookie* について
- 6.1.3.3. *Clipboard* へのアクセス通知について

下記の構成・内容を見直し拡充いたしました

- 4.1.3.1. *exported* 設定と *intent-filter* 設定の組み合わせ *(Activity* の場合*)*
- 4.9.2.5. *Intent* 経由など、他から受け取った *URL* が想定された *URL* か確認する（必須）
- 5.5.3.5. データアクセスの監査について
- 5.5.3.6. 位置情報アクセスについて
- 5.6.3.5. *Conscrypt* モジュールについて
- 6.1.3.2. *Clipboard* に格納されている情報の操作

2022-1-17

下記の構成・内容を見直し拡充いたしました

- 2.2. サンプルコード、ルールブック、アドバンスト

2022-8-29

下記の新しい記事を追加いたしました

- 4.7.3.3. *Intent Filter* の厳格化
- 4.2.3.8. 動的 *Broadcast Receiver* の安全性の強化
- 4.10.3.3. 通知に関する実行時の権限
- 5.2.3.8. 実行時の権限の取り消し
- 4.11.4.2. 新規インストールアプリでの *sharedUserId* の無効化

下記の構成・内容を見直し拡充いたしました

- 2.4. *Android* セキュアコーディング関連書籍の紹介
- 2.5. サンプルコードの *Android Studio* への取り込み手順
- 5.5.3.6. 位置情報アクセスについて
- 6.1.3.2. *Clipboard* に格納されている情報の操作

2024-2-29

下記の新しい記事を追加いたしました

- 4.4.3.3. 必須となった *Service* タイプ指定
- 4.6.3.7. *DCL (Dynamic Code Loading)* の安全性の強化

- 4.6.3.9. *Zip* ファイルの *Path Traversal* 対策
- 4.10.3.4. 進行中を示す *Notification* の動作の変更
- 4.6.3.8. メディア所有者のパッケージ名

下記の構成・内容を見直し拡充いたしました

- 4.1.3.3. ルート *Activity* について
- 4.2.3.8. 動的 *Broadcast Receiver* の安全性の強化

2025-1-29

下記の新しい記事を追加いたしました

- 4.7.3.4. *Intent Filter* 機能の拡張
- 4.7.3.5. *PendingIntent* とパッケージ停止状態の仕様変更
- 4.3.3.1. コンテンツ *URI* の権限管理
- 4.4.3.4. フォアグラウンド サービスの変更
- 4.6.3.5. *Android 10* 以降の外部ストレージアクセス
- 4.6.3.6. *Android 10* 以降の共有ストレージアクセス
- 4.6.3.10. 選択した写真へのアクセスについて、直近のユーザーの選択をクエリする
- 5.5.3.14. プライベート スペース
- 5.1.3.5. 認証情報マネージャーを自動入力と統合する
- 5.1.3.4. 画面録画の検出
- 5.5.3.12. 部分的画面共有
- 5.5.3.13. *DND* モードのグローバル状態管理の変更点
- 5.6.3.6. バックアップデータの漏洩対策
- 5.6.3.7. アプリ内ハードコーディングのセキュリティリスク

下記の記事を削除致しました

- 4.6.3.6. Android 10(API Level 29) における外部ストレージへのアクセスに関する仕様について
- 4.6.3.7. Android 11（API Level 30）における対象範囲別ストレージの適用について
- 4.6.3.8. Android 13（API Level 33）におけるメディアコレクション権限について
- 4.6.3.9. Android 14（API Level 34）における画像と動画の部分的アクセス

2025-8-27

下記の新しい記事を追加いたしました

- 4.2.3.9. 順序付き *Broadcast* の *priority* 仕様変更
- 4.6.3.11. *MediaStore* バージョンのロックダウン

- 4.6.3.12. *App-owned photos*
- 4.7.3.6. *Safer Intents*
- 4.7.3.7. *Intent* リダイレクト攻撃対策
- 5.5.3.7. ローカル ネットワークへのアクセス権
- 5.6.3.8. *Key sharing API* によるアプリ間鍵共有の仕組みとリスク
- 5.6.3.9. （コラム） *Google Play* アプリ署名による鍵管理

下記の構成・内容を見直し拡充いたしました

- 4.6.3.4. *Android 10* 以降の内部ストレージアクセス
- 4.6.3.5. *Android 10* 以降の外部ストレージアクセス
- 4.6.3.6. *Android 10* 以降の共有ストレージアクセス
- 4.7.3.1. コンポーネント公開制御と *Intent* 送信の制限
- 4.7.3.2. *PendingIntent* オブジェクトの可変性
- 4.7.3.3. *Intent Filter* の厳格化
- 4.7.3.4. *Intent Filter* 機能の拡張
- 4.7.3.5. *PendingIntent* とパッケージ停止状態の仕様変更
- 5.5.3.14. プライベート スペース
- 5.5.3.8. *Android 12* における使用していないアプリの自動休止機能について
- 5.5.3.9. パッケージアクセスの仕様変更にともなう *API* の戻り値の変化
- 4.11.4.2. 新規インストールアプリでの *sharedUserId* の無効化
- 2.2. サンプルコード、ルールブック、アドバンスト
- 5.1.3.5. 認証情報マネージャーを自動入力と統合する
- 4.6.3.8. メディア所有者のパッケージ名
- 5.3.3.2. *Android 8.0*（*API Level 26*）以降で署名の一致しない *Authenticator* のアカウントを読めるケース

下記の記事を削除致しました

- 4.1.3.3. Activity に送信される Intent の読み取り (Android 5.0 より前のバージョンについて)
- 4.6.3.4. Android 4.4 (API Level 19) における外部ストレージへのアクセスに関する仕様変更について
- 4.6.3.5. Android 7.0（API Level 24）における外部ストレージの特定ディレクトリへのアクセスに関する仕様変更について
- 4.9.3.1. Android 4.2 未満の端末における addJavascriptInterface() に起因する脆弱性について
- 5.3.3.2. Android 4.0.x では利用アプリと Authenticator アプリの署名鍵が異なると例外が発生する
- 5.6.3.3. 乱数生成における脆弱性と対策

2026-9-16

下記の新しい記事を追加いたしました

- 4.1.3.7. バックグラウンドアクティビティ起動（*BAL*）の制限強化
- 4.2.3.10. *Bluetooth* ボンドの損失に対する自律的な再ペア設定
- 4.3.3.2. 連絡先プロバイダ（*CP2*）の厳格化
- 4.4.3.5. バックグラウンドオーディオの制限強化
- 4.7.3.8. 暗黙的な *URI* 権限付与の制限
- 4.7.3.9. プロファイル間のループバック制限
- 4.12. *NFC*・*FeliCa* を利用する
- 5.2.3.9. *SMS OTP*（ワンタイムパスワード）の保護強化
- 5.4.3.9. *ECH*（*Encrypted Client Hello*）の有効化
- 5.4.3.10. 証明書の透明性（*CT*）への対応
- 5.4.3.11. *usesCleartextTraffic* の非推奨化と *HTTP* 通信の許可
- 5.6.3.10. アプリごとの *Android Keystore*（鍵）上限

下記の構成・内容を見直し拡充いたしました

- 4.6.3.7. *DCL (Dynamic Code Loading)* の安全性の強化
- 5.1. 認証（パスワード・パスキー）画面を作る
- 5.4.1.1. *HTTP* 通信する
- 5.4.1.2. *HTTPS* 通信する
- 5.5.3.6. 位置情報アクセスについて
- 5.5.3.7. ローカル ネットワークへのアクセス権

下記の記事を削除いたしました

- 4.1.3.7. 安全なバックグラウンドアクティビティの起動

※ 改訂内容の詳細については「1.4. *2025* 年 *8* 月 *27* 日版からの訂正記事について」を参照して下さい。

新版の公開にあたり、皆様から頂いたご意見・コメントを元に本ガイドの内容を更新しました。

制作

一般社団法人 日本スマートフォンセキュリティ協会 技術部会 セキュアコーディング WG

リーダー	宮崎 力	株式会社ラック
メンバー	仲上 竜太	株式会社ラック

次のページに続く

表 6.1.1 – 前のページからの続き

塩田 明弘	株式会社 NTT データグループ
本間 輝彰	KDDI 株式会社
上松 晴信	KDDI 株式会社

（執筆関係者、社名五十音順）

2025 年 8 月 27 日版制作者

リーダー

宮崎力 株式会社ラック

メンバー

仲上 竜太	株式会社ラック
塩田 明弘	株式会社 NTT データグループ
本間 輝彰	KDDI 株式会社
上松 晴信	KDDI 株式会社

（執筆関係者、社名五十音順）

2025 年 1 月 29 日版制作者

リーダー

宮崎力 株式会社ラック

メンバー

仲上 竜太	ニューリジェンセキュリティ株式会社
塩田 明弘	株式会社 NTT データグループ
本間 輝彰	KDDI 株式会社
上松 晴信	KDDI 株式会社

（執筆関係者、社名五十音順）

2024 年 **2** 月 **29** 日版制作者

リーダー

宮崎力 株式会社ラック

メンバー

塩田 明弘	株式会社 NTT データグループ
本間 輝彰	KDDI 株式会社
上松 晴信	KDDI 株式会社
小笠原 徳彦	株式会社 SHIFT SECURITY
才田 好則	日本電気株式会社
青柳 亨	日本電気株式会社

（執筆関係者、社名五十音順）

2022 年 8 月 29 日版制作者

リーダー

宮崎力 株式会社ラック

メンバー

Pantuhong Sorasiri	株式会社ラック
塩田 明弘	株式会社エヌ・ティ・ティ・データ
本間 輝彰	KDDI 株式会社
上松 晴信	KDDI 株式会社

（執筆関係者、社名五十音順）

2022 年 1 月 17 日版制作者

リーダー

宮崎力 株式会社ラック

メンバー

本間 輝彰	KDDI 株式会社

（執筆関係者、社名五十音順）

2021 年 10 月 19 日版制作者

リーダー

宮崎力 株式会社ラック

メンバー

小松 奈央	株式会社ラック
本間 輝彰	KDDI 株式会社

（執筆関係者、社名五十音順）

2020 年 11 月 1 日版制作者

リーダー

宮崎力 株式会社ラック

メンバー

塩田 明弘	株式会社エヌ・ティ・ティ・データ
本間 輝彰	KDDI 株式会社
才田 好則	日本電気株式会社

（執筆関係者、社名五十音順）

2019 年 9 月 1 日版制作者

リーダー

小木曽純 ソニーデジタルネットワークアプリケーションズ株式会社

メンバー

澤田 壽實、鈴木 康平	株式会社 SRA
塩田 明弘	株式会社エヌ・ティ・ティ・データ
本間 輝彰	KDDI 株式会社
山口 信明	ソニーデジタルネットワークアプリケーションズ株式会社
谷口 岳	タオソフトウェア株式会社
伊藤 健文	日本システム開発株式会社

（執筆関係者、社名五十音順）

2018 年 9 月 1 日版制作者

リーダー

安藤彰 ソニーデジタルネットワークアプリケーションズ株式会社

メンバー

澤田 壽實、鈴木 康平	株式会社 SRA
本間 輝彰	KDDI 株式会社
小木曽 純、久本 純樹、山口 信明	ソニーデジタルネットワークアプリケーションズ株式会社
谷田部 茂	株式会社フォーマルハウト・テクノ・ソリューションズ

（執筆関係者、社名五十音順）

2018 年 2 月 1 日版制作者

リーダー

安藤彰 ソニーデジタルネットワークアプリケーションズ株式会社

メンバー

奥山 謙	Android セキュリティ部
星本 英史	株式会社 SRA
武井 滋紀	エヌ・ティ・ティ・ソフトウェア株式会社
塩田 明弘	株式会社エヌ・ティ・ティ・データ
福本 郁哉	一般社団法人 JPCERT コーディネーションセンター (JPCERT/CC)
小木曽 純、久本 純樹、山口 信明、吉田 万里子	ソニーデジタルネットワークアプリケーションズ株式会社
笠原 正弘	ソフトバンクモバイル株式会社
伊藤 健文	日本システム開発株式会社
谷田部 茂	株式会社フォーマルハウト・テクノ・ソリューションズ

(執筆関係者、社名五十音順)

2017 年 2 月 1 日版制作者

リーダー

奥山謙 ソニーデジタルネットワークアプリケーションズ株式会社

メンバー

荒木 成治、島野 英司	Android セキュリティ部
大内 智美、福本 郁哉、山野井 陽一	株式会社 SRA
武井 滋紀	エヌ・ティ・ティ・ソフトウェア株式会社
塩田 明弘	株式会社エヌ・ティ・ティ・データ
高橋 哲也	株式会社スクウェア・エニックス
山地 秀典	ソニー株式会社
安藤 彰、小木曽 純、松並 勝	ソニーデジタルネットワークアプリケーションズ株式会社
谷口 岳	タオソフトウェア株式会社

(執筆関係者、社名五十音順)

2016 年 9 月 1 日版制作者

リーダー

松並勝 ソニーデジタルネットワークアプリケーションズ株式会社

メンバー

荒木 成治	Android セキュリティ部
大内 智美、福本 郁哉	株式会社 SRA
武井 滋紀	エヌ・ティ・ティ・ソフトウェア株式会社
大園 通	シスコシステムズ合同会社
山地 秀典	ソニー株式会社
安藤 彰、大谷 三岳、小木曽 純、奥山 謙	ソニーデジタルネットワークアプリケーションズ株式会社
島野 英司、谷口 岳	タオソフトウェア株式会社
満園 大祐	日本システム開発株式会社

(執筆関係者、社名五十音順)

2016 年 2 月 1 日版制作者

リーダー

松並勝 ソニーデジタルネットワークアプリケーションズ株式会社

メンバー

安達 正臣	Android セキュリティ部
福本 郁哉、星本 英史	株式会社 SRA
武井 滋紀	エヌ・ティ・ティ・ソフトウェア株式会社
大園 通	シスコシステムズ合同会社
安藤 彰、伊藤 妙子、大谷 三岳、奥山 謙、楫節子、西村 宗晃	ソニーデジタルネットワークアプリケーションズ株式会社
山地 秀典	ソニーモバイルコミュニケーションズ株式会社
笠原 正弘	ソフトバンクモバイル株式会社
島野 英司、谷口 岳	タオソフトウェア株式会社

(執筆関係者、社名五十音順)

2015 年 6 月 1 日版制作者

リーダー

松並勝 ソニーデジタルネットワークアプリケーションズ株式会社

メンバー

星本 英史	株式会社 SRA
武井 滋紀	エヌ・ティ・ティ・ソフトウェア株式会社
大園 通	シスコシステムズ合同会社
安藤 彰、奥山 謙、西村 宗晃	ソニーデジタルネットワークアプリケーションズ株式会社
笠原 正弘	ソフトバンクモバイル株式会社
島野 英司、谷口 岳	タオソフトウェア株式会社
八津川 直伸	日本ユニシス株式会社
谷田部 茂	株式会社フォーマルハウト・テクノ・ソリューションズ
今西 杏丞、河原 豊、近藤 昭雄、志村 直彦、新谷 正人、原 昇平、藤澤 智之、藤田 竜史、三竹 一馬	株式会社ブリリアントサービス

(執筆関係者、社名五十音順)

2014 年 7 月 1 日版制作者

リーダー

松並勝 ソニーデジタルネットワークアプリケーションズ株式会社

メンバー

熊澤 努、星本 英史	株式会社 SRA
武井 滋紀	エヌ・ティ・ティ・ソフトウェア株式会社
竹森 敬祐、磯原 隆将	KDDI 株式会社
大園 通	シスコシステムズ合同会社
安藤 彰、伊藤 妙子、奥山 謙、楫 節子、片岡 良典	ソニーデジタルネットワークアプリケーションズ株式会社
笠原 正弘	ソフトバンクモバイル株式会社
島野 英司、谷口 岳	タオソフトウェア株式会社
佐藤 導吉	東京システムハウス株式会社
八津川 直伸	日本ユニシス株式会社
谷田部 茂	株式会社フォーマルハウト・テクノ・ソリューションズ

(執筆関係者、社名五十音順)

2013 年 4 月 1 日版制作者

リーダー

松並勝 ソニーデジタルネットワークアプリケーションズ株式会社

メンバー

安達 正臣、長谷川 智之	Android セキュリティ部
安部 勇気、大内 智美、熊澤 努、澤田 寿実、畑 清志、比嘉 陽一、福井 悠、福本 郁哉、星本 英史、横井 俊、吉澤 孝和	株式会社 SRA
藤原 健	NRI セキュアテクノロジーズ株式会社
武井 滋紀	エヌ・ティ・ティ・ソフトウェア株式会社
竹森 敬祐	KDDI 株式会社
久保 正樹、熊谷 裕志、戸田 洋三	一般社団法人 JPCERT コーディネーションセンター (JPCERT/CC)
大園 通	シスコシステムズ合同会社
新井 幹也、坂本 昌彦	株式会社セキュアスカイ・テクノロジー
浅野 徹、安藤 彰、池邉 亮志、小木曽 純、奥山 謙、片岡 良典、西村 宗晃、古澤 浩司、山岡 研二	ソニーデジタルネットワークアプリケーションズ株式会社
谷口 岳	タオソフトウェア株式会社
八津川 直伸	日本ユニシス株式会社
谷田部 茂	株式会社フォーマルハウト・テクノ・ソリューションズ

(執筆関係者、社名五十音順)

2012 年 11 月 1 日版制作者

リーダー

松並勝 ソニーデジタルネットワークアプリケーションズ株式会社

メンバー

佐藤 勝彦、中口 明彦	Android セキュリティ部
大内 智美、大平 直之、熊澤 努、関川 未来、中野 正剛、比嘉 陽一、福本 郁哉、星本 英史、安田 章一、八尋 唯行、吉澤 孝和	株式会社 SRA
武井 滋紀	エヌ・ティ・ティ・ソフトウェア株式会社
竹森 敬祐	KDDI 株式会社
久保 正樹、熊谷 裕志、戸田 洋三	一般社団法人 JPCERT コーディネーションセンター (JPCERT/CC)
大園 通	シスコシステムズ合同会社
浅野 徹、安藤 彰、池邉 亮志、市川 茂、大谷 三岳、小木曽 純、奥山 謙、片岡 良典、佐藤 郁恵、西村 宗晃、山岡 一夫、吉川 岳流	ソニーデジタルネットワークアプリケーションズ株式会社
谷口 岳、島野 英司、北村 久雄	タオソフトウェア株式会社
山川 隆郎	一般社団法人日本オンラインゲーム協会
石原 正樹、森 靖晃	日本システム開発株式会社
八津川 直伸	日本ユニシス株式会社
谷田部 茂	株式会社フォーマルハウト・テクノ・ソリューションズ
藤井 茂樹	ユニアデックス株式会社

(執筆関係者、社名五十音順)

2012 年 6 月 1 日版制作者

リーダー

松並勝 ソニーデジタルネットワークアプリケーションズ株式会社

メンバー

佐藤 勝彦	Android セキュリティ部
大内 智美、比嘉 陽一、星本 英史	株式会社 SRA
武井 滋紀	エヌ・ティ・ティ・ソフトウェア株式会社
千田 雅明	グリー株式会社
久保 正樹、熊谷 裕志、戸田 洋三	一般社団法人 JPCERT コーディネーションセンター (JPCERT/CC)
大園 通、谷田部 茂	シスコシステムズ合同会社
田口 陽一	株式会社システムハウス．アイエヌジー
坂本 昌彦	株式会社セキュアスカイ・テクノロジー
安藤 彰、市川 茂、奥山 謙、佐藤 郁恵、西村 宗晃、山岡 一夫	ソニーデジタルネットワークアプリケーションズ株式会社
谷口 岳、島野 英司、北村 久雄	タオソフトウェア株式会社
佐藤 導吉	東京システムハウス株式会社
服部 正和	トレンドマイクロ株式会社
八津川 直伸	日本ユニシス株式会社
谷田部 茂	株式会社フォーマルハウト・テクノ・ソリューションズ
藤井 茂樹	ユニアデックス株式会社

(執筆関係者、社名五十音順)