# Androidアプリと鍵管理

## – これからの鍵管理の話をしよう -

Sony Digital Network Applications, Inc.
安藤　彰
2016/03/23　JSSEC セキュアコーディングディ

# 自己紹介

- 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)

- 1996年 ソニー(株) 入社
  - 主にソフトウェア開発業務
- 2006年より現籍
- 最近の業務
  - DRM モジュール開発(セキュア設計・耐タンパ実装)
  - セキュリティ(設計・分析・提案)コンサルティング
  - セキュリティツール開発
- JSSEC
  - 初版(2012年)より JSSEC セキュアコーディングガイド執筆活動に参画

# Agenda

- 鍵管理のおさらい
  - Android アプリ編
- Android 6.0 の鍵管理機能
  - AndroidKeyStore
- Android N の追加機能

# 鍵管理のおさらい

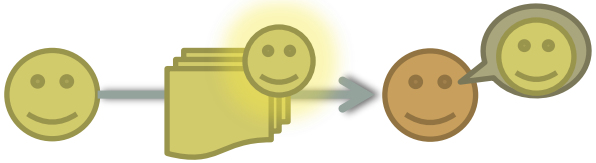
# 暗号技術って何のために必要？

- 暗号技術の目的
  - 秘匿性の確保
    - 暗号化(共通鍵、公開鍵)
  - 完全性の確保
  - 認証
    - デジタル署名、MAC (メッセージ認証コード)
  - 否認防止
    - デジタル署名

| | 暗号化 | デジタル署名 (PKI) | MAC |
|---|---|---|---|
| 秘匿性の確保 | ○ | × | × |
| 完全性の確保 | × | ○ | ○ |
| 認証 | × | ○ | ○ |
| 否認防止 | × | ○ | × |

# 暗号技術使ってますか？

- 用途
  - 大事な情報の漏えいを防ぎたい
    - 暗号化
  - 通信相手に正しい情報を伝えたい(改ざん検知)
    - デジタル署名、MAC
  - 正しい持ち主(ユーザー、アプリなど)であることを証明したい
    - デジタル署名、MAC
- 身近では、
  - HTTPS 通信、VPN、WiFi、DRM、ディスクの暗号化、暗号化 ZIP、etc...



# 使う際にはどんなことに気を付ける？

- 暗号技術を使う時の注意事項
  - 正しいアルゴリズムを使用する

例えば、暗号化の場合
アルゴリズム　　：AES
ブロック暗号モード：CBC
パッディング　　：PKCS#7
鍵長　　：128 ビット

  - 鍵を安全に扱う

より重要

今日のテーマはこちら

# (参考) 正しいアルゴリズムを使用する

NIST（米国） NIST SP800-57

| アルゴリズムのライフタイム | 対称鍵暗号 | 非対称暗号 | 楕円暗号 | HASH (デジタル署名, HASH) | HASH (HMAC,KD, 乱数生成) |
|---|---|---|---|---|---|
| ～2010 | 80 | 1024 | 160 | 160 | 160 |
| ～2030 | 112 | 2048 | 224 | 224 | 160 |
| 2030～ | 128 | 3072 | 256 | 256 | 160 |

ECRYPT II (EU)

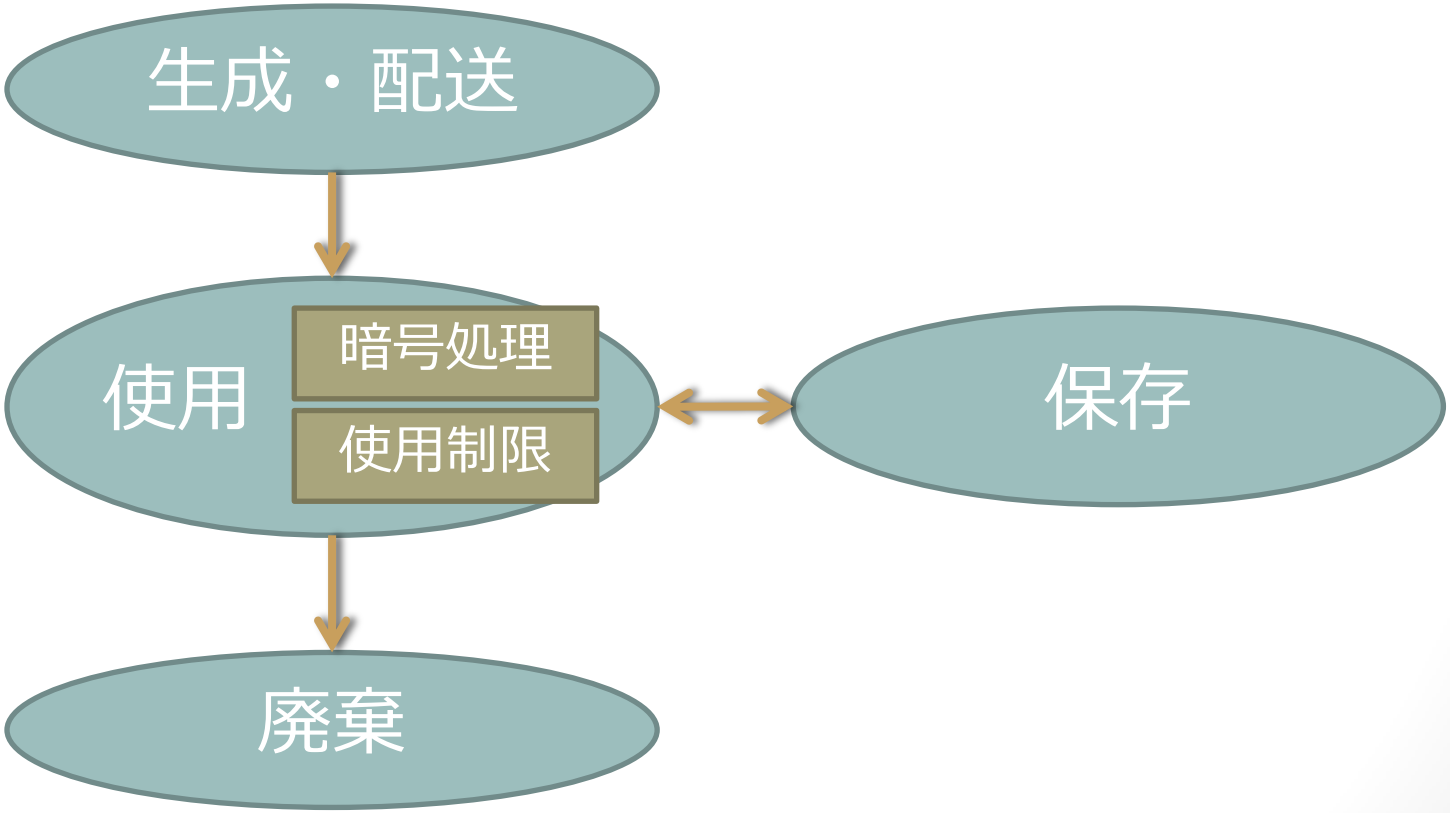
| アルゴリズムのライフタイム | 対称鍵暗号 | 非対称暗号 | 楕円暗号 | HASH |
|---|---|---|---|---|
| 2009～2012 | 80 | 1248 | 160 | 160 |
| 2009～2020 | 96 | 1776 | 192 | 192 |
| 2009～2030 | 112 | 2432 | 224 | 224 |
| 2009～2040 | 128 | 3248 | 256 | 256 |
| 2009～ | 256 | 15424 | 512 | 512 |

CRYPTREC(日本) 電子政府推奨暗号リスト

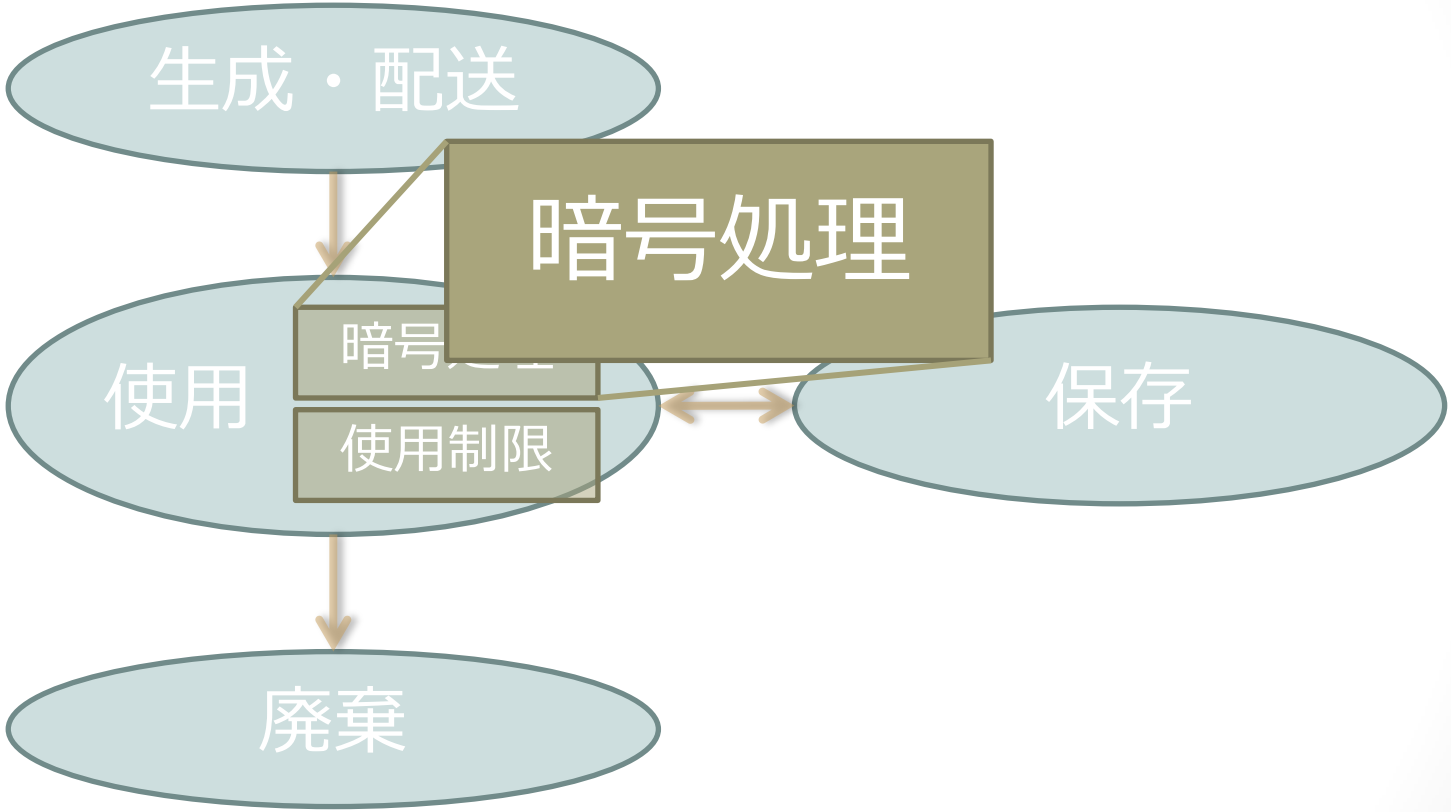
| 技術分類 | | 名称 |
|---|---|---|
| 公開鍵暗号 | 署名 | DSA,ECDSA,RSA-PSS,RSASSA-PKCS1-V1_5 |
| | 守秘 | RSA-OAEP |
| | 鍵共有 | DH,ECDH |
| 共通鍵暗号 | 64ビットブロック暗号 | 3-key Triple DES |
| | 128ビットブロック暗号 | AES,Camellia |
| | ストリーム暗号 | KCipher-2 |
| ハッシュ関数 | | SHA-256,SHA-384,SHA-512 |
| 暗号利用モード | 秘匿モード | CBC,CFB,CTR,OFB |
| | 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 |

# 鍵を安全に扱う

- 鍵のライフサイクルにおいて、各フェーズで安全に扱う必要がある



# どこで使用するか？(暗号処理)



# どこで使用するか？(暗号処理)

## Normal World

一般アプリプロセス

システムプロセス

## SW 耐タンパ

難読化、アンチデバッグ、ホワイトボックス暗号 etc...

プロセス

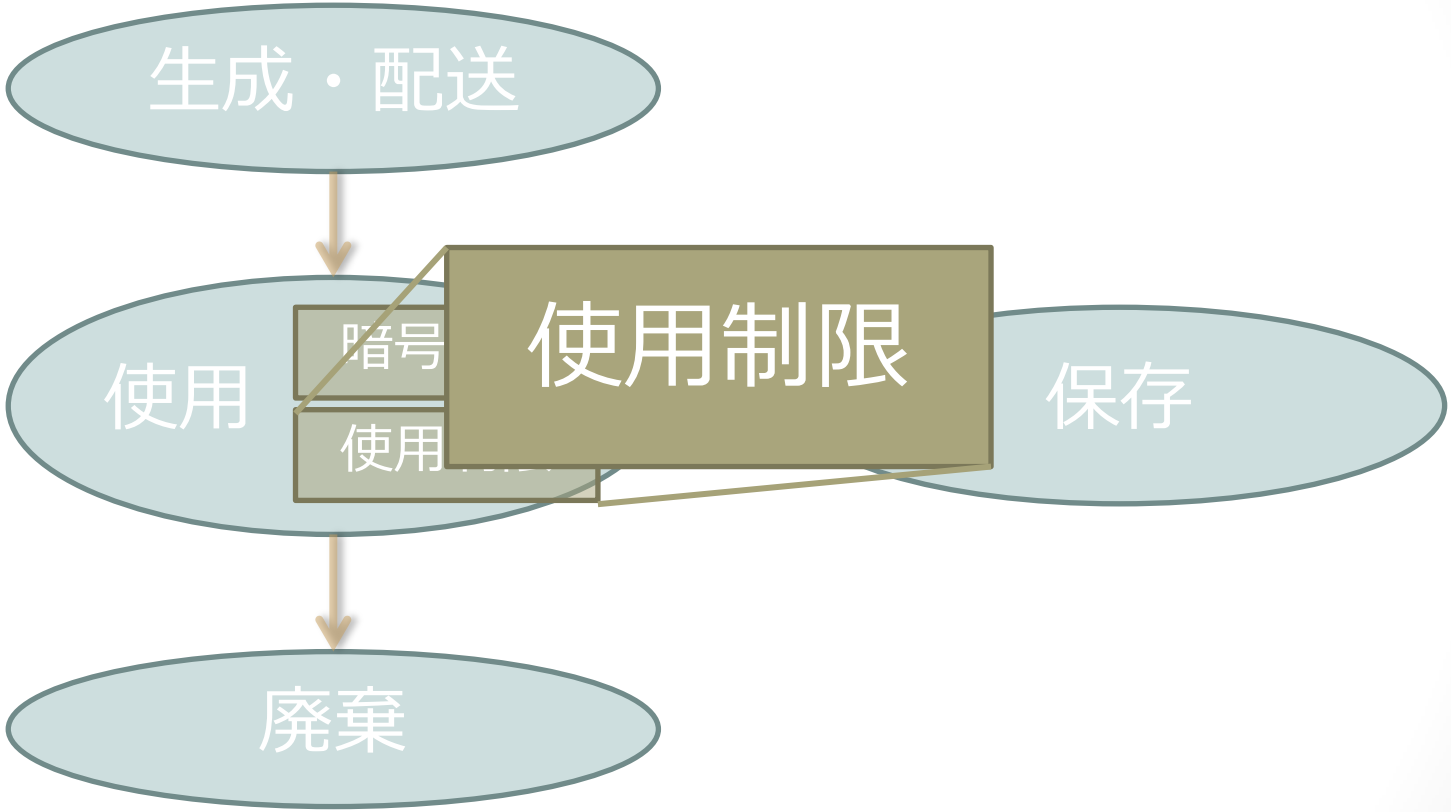
SW 耐タンパ領域

## Secure World (HW 耐タンパ)

TEE (Trusted Execution Environment)
TrustZone, Intel SGX, etc…
TPM (Trusted Platform Module)

セキュアプロセス

# 誰が使用するか？(使用制限)

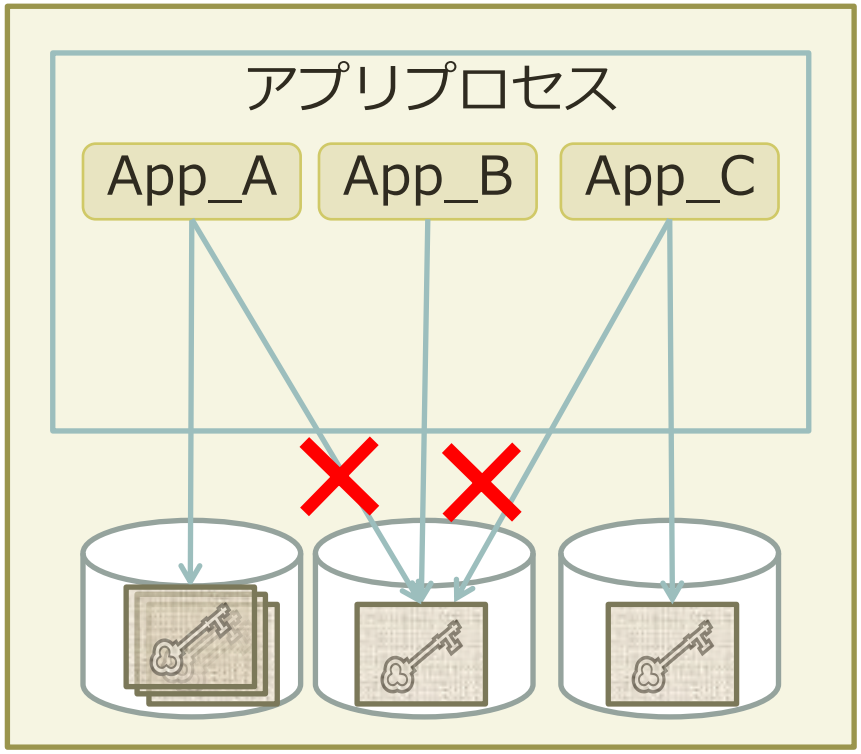


# 誰が使用するか？(使用制限)

- 鍵の使用者を制限する
  - アプリ単位
    - 最も一般的。鍵を管理するアプリだけが鍵を使用することができる
  - 機能単位
    - システムの提供する機能が鍵を使用することができる
  - ユーザー単位
    - 特定のユーザーに限り鍵を使用することができる

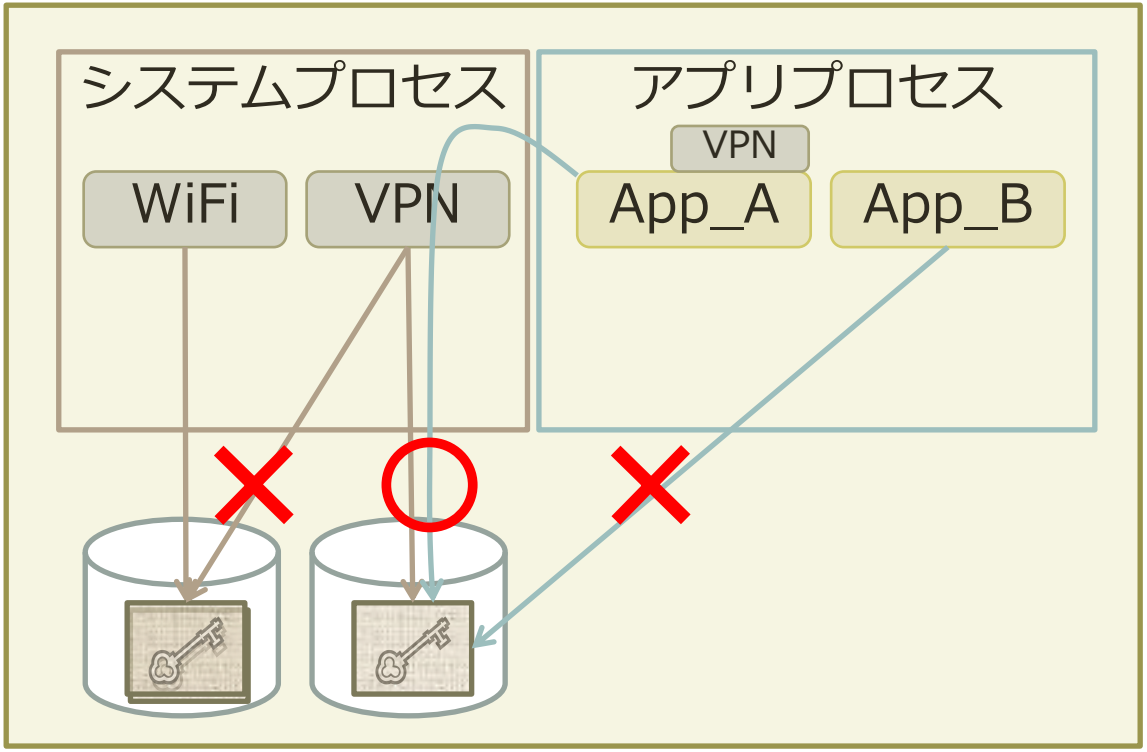
# 誰が使用するか？(使用制限)
## - アプリ単位 -

- 最も一般的。鍵を管理するアプリだけが鍵を使用することができる



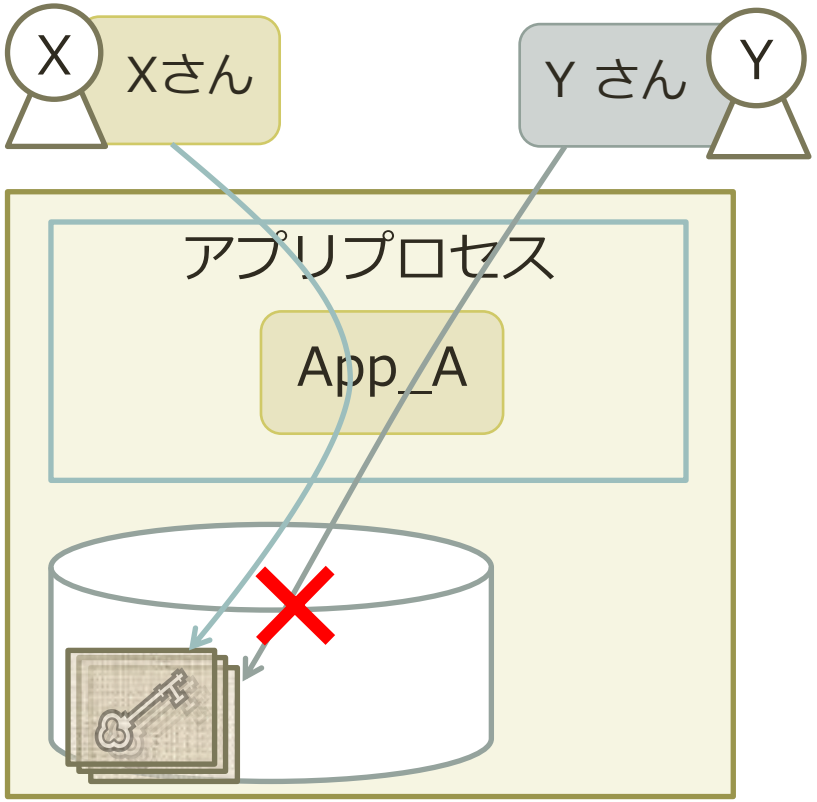
# 誰が使用するか？(使用制限)
## - 機能単位 -

- システムの提供する機能が鍵を使用することができる

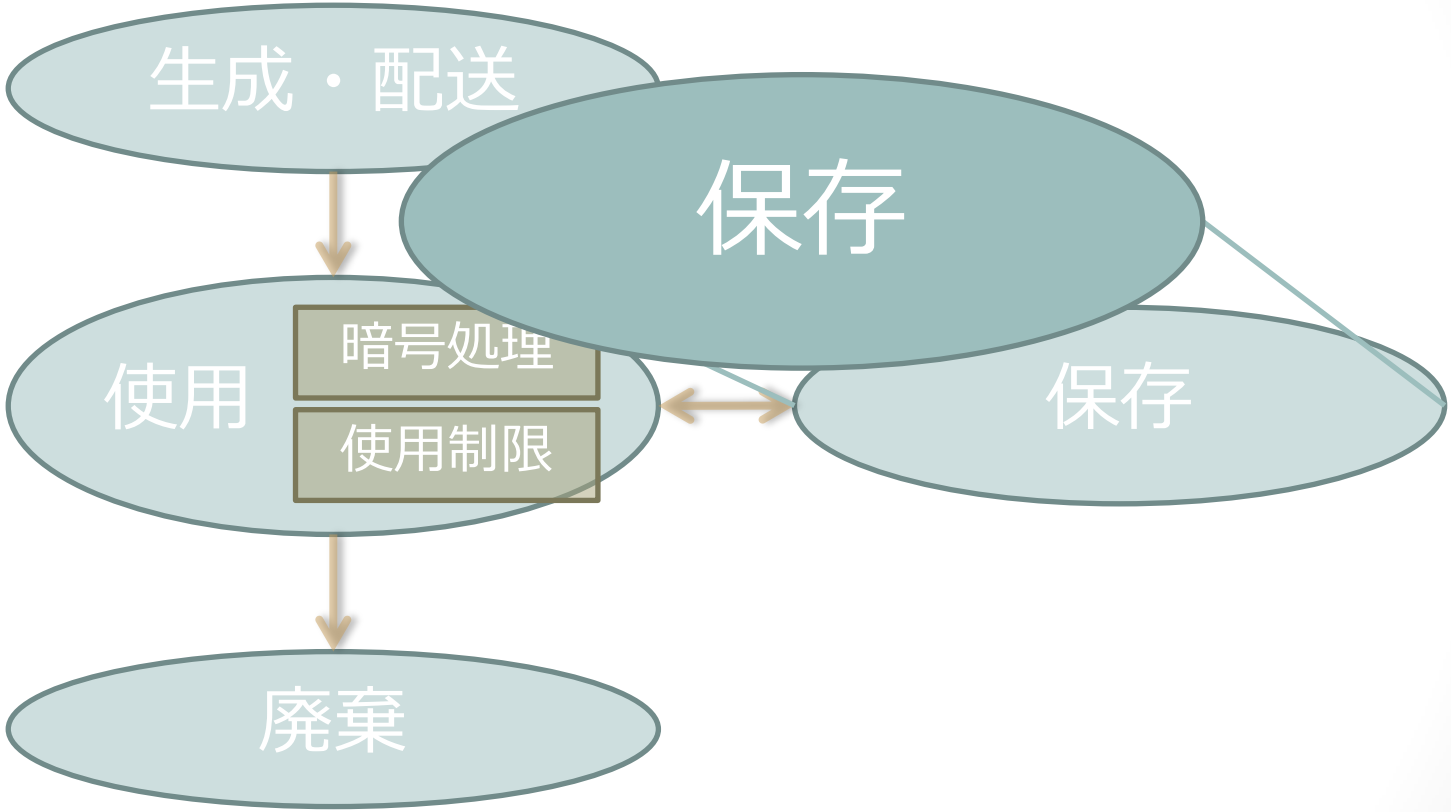


# 誰が使用するか？(使用制限)
## - ユーザー単位 -

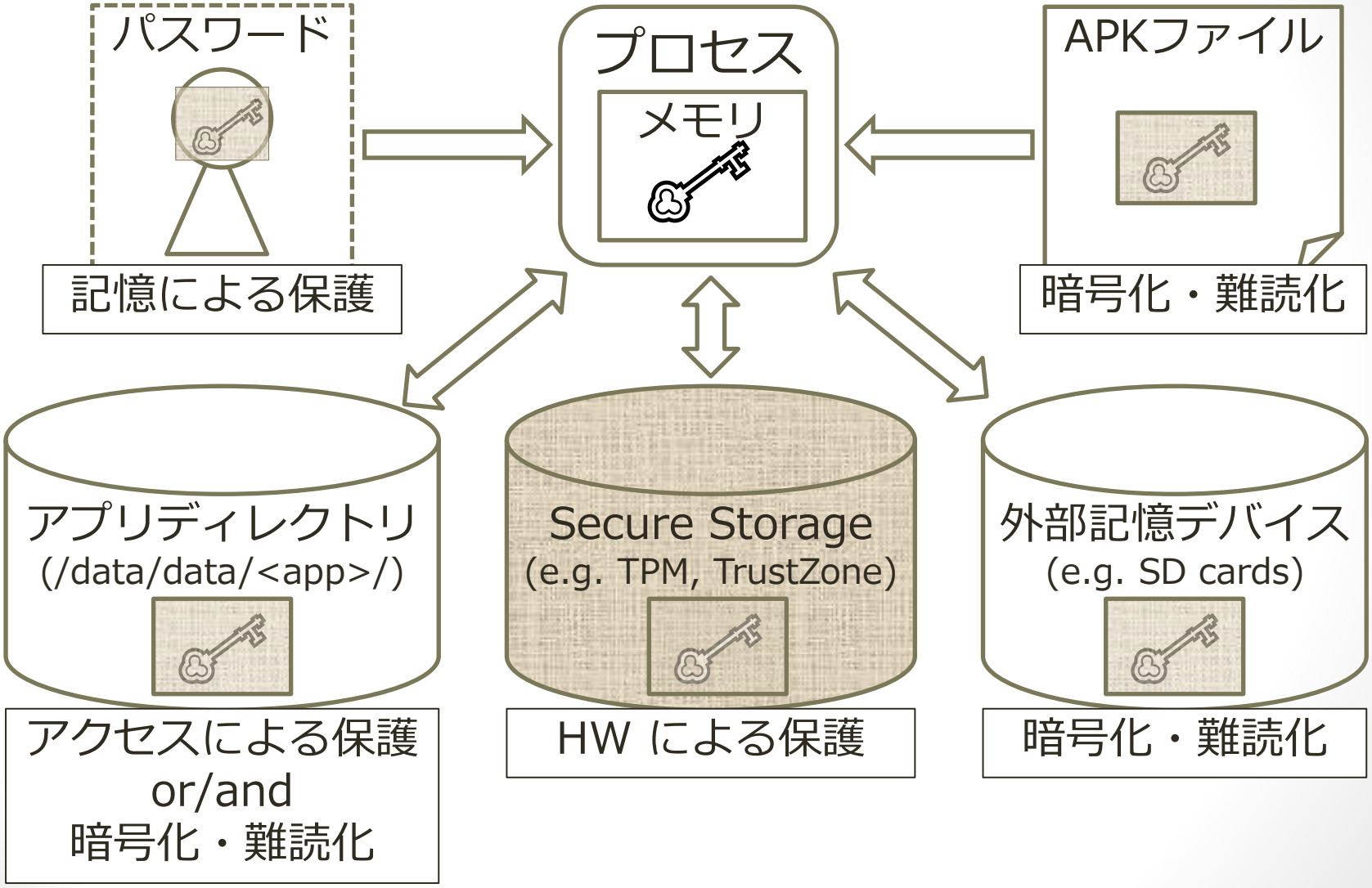
- 特定のユーザーに限り鍵を使用することができる



# どこに保存するか？

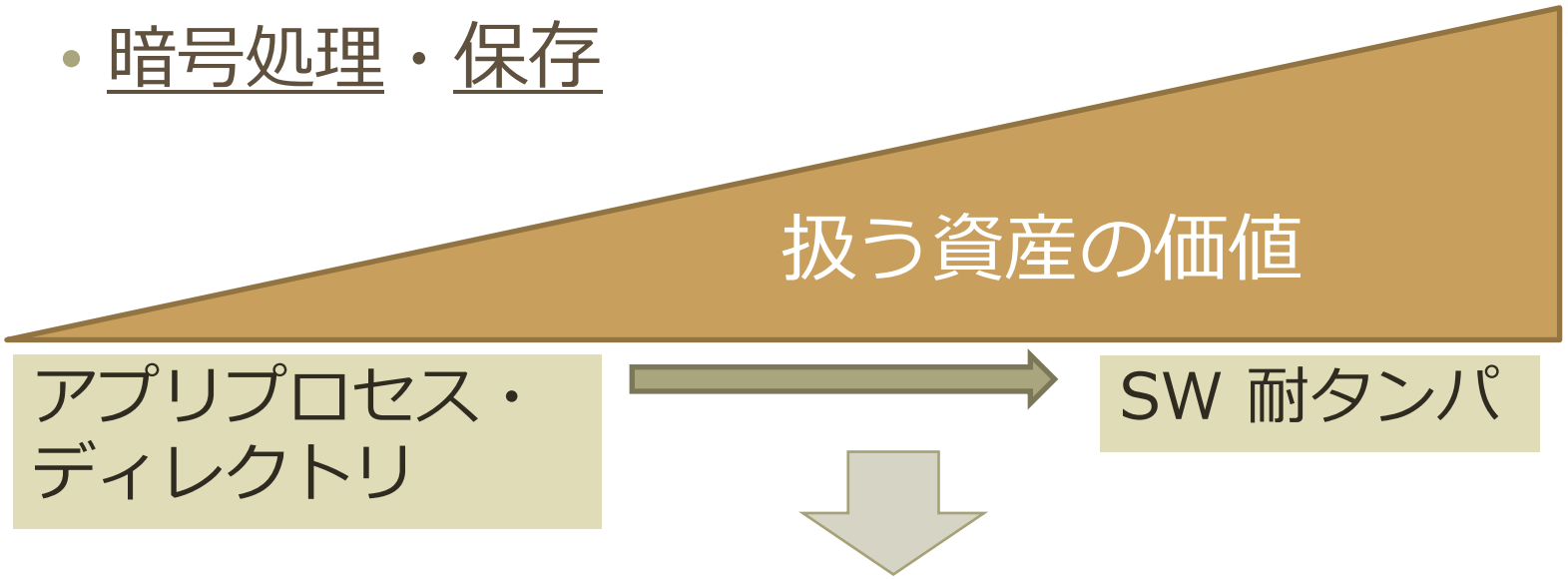


# どこに保存するか？



# Android アプリに当てはめると？

- 暗号処理・保存

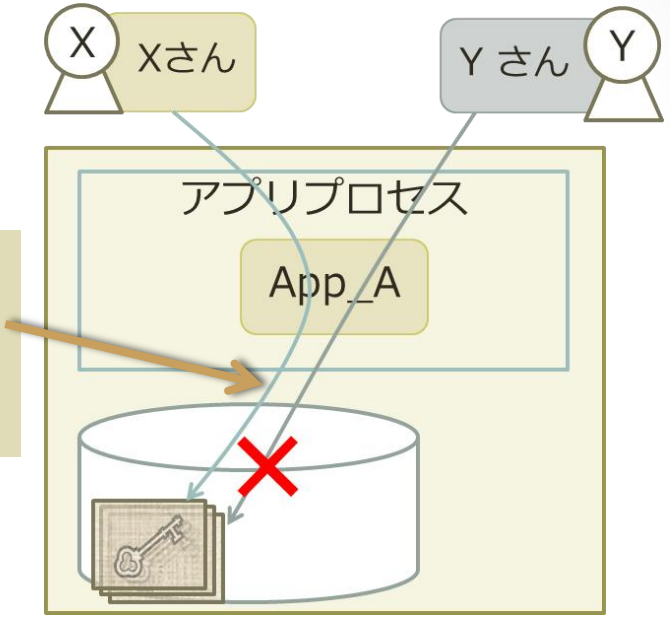



境目はどこ？
やはり費用も気になる

# Android アプリに当てはめると？

- 使用制限

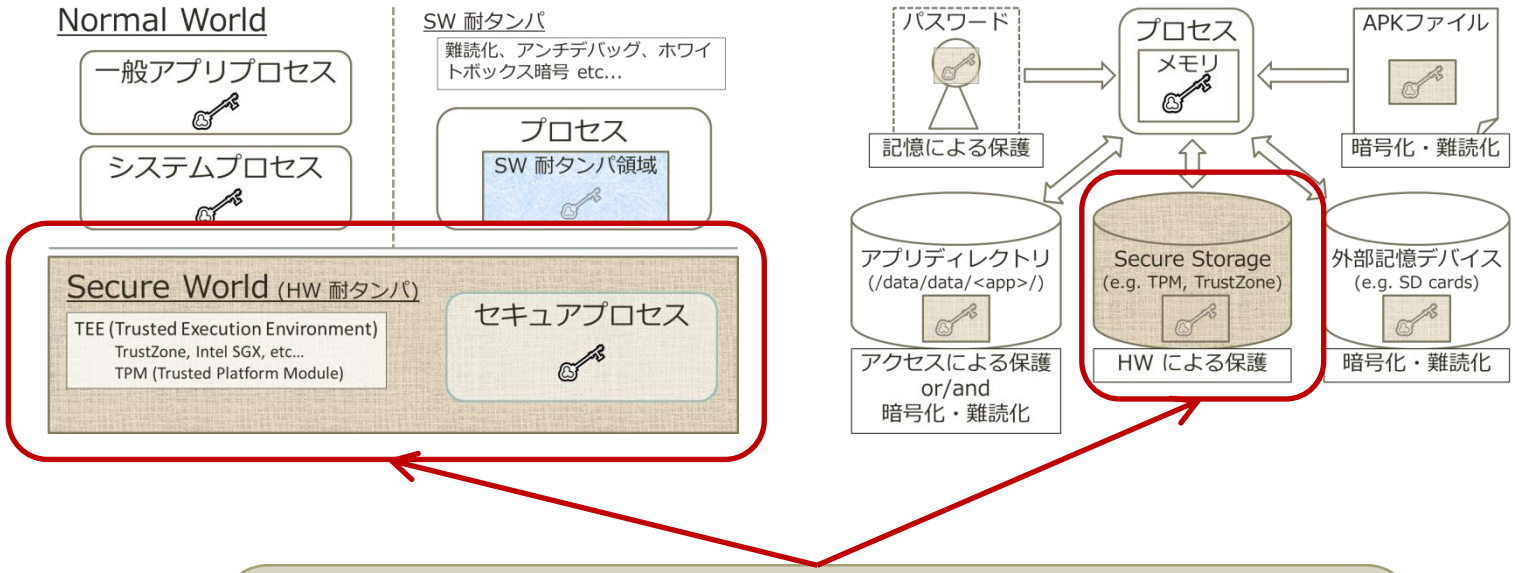
ユーザー認証どうします？




やっぱパスワードかな？
実装大丈夫かな？

# Android アプリに当てはめると？

- HW 保護機能




ハードウェアによる保護って一般アプリで使えないの？

# こんなニュースも(昨日)

- ルート化マルウェアのリスクもゼロではない！！

グーグル、「Android」端末向け緊急パッチをリリース

Liam Tung （Special to ZDNet.com） 翻訳校正： 編集部　2016/03/22 11:00

いいね！ 18　ツイート

印刷　メール

PR | 基幹データとビッグデー

Googleがパッチ未適
り組んでいる。このバ

Googleは大半のAnd
「Nexus」製品群向け
化アプリが複数存在す

米国時間3月18日に出されたアドバイザリによると、名前の明かされていないそれらのルート化アプリ（「Google Play」やそのほかのアプリストアで公開されている）が原因で、「端末がローカルで永続的な危険にさらされる」おそれがあるという。

# Android アプリに当てはめると？

ご安心ください！！

# Androidアプリに当てはめると？

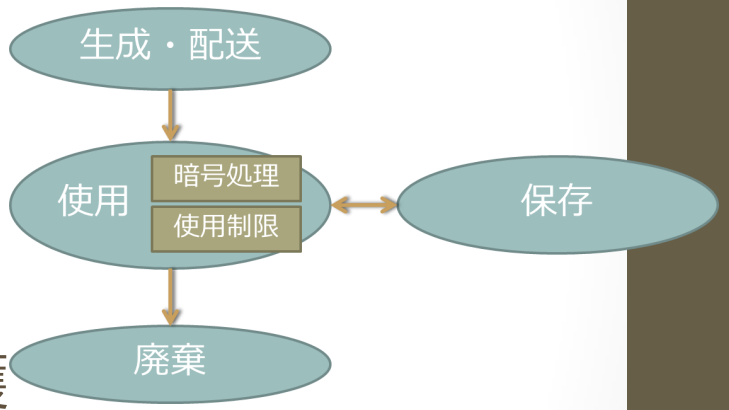
# AndroidKeyStore
## がありますよ！

AndroidKeyStore

# ANDROID 6.0 の鍵管理機能

# AndroidKeyStore とは

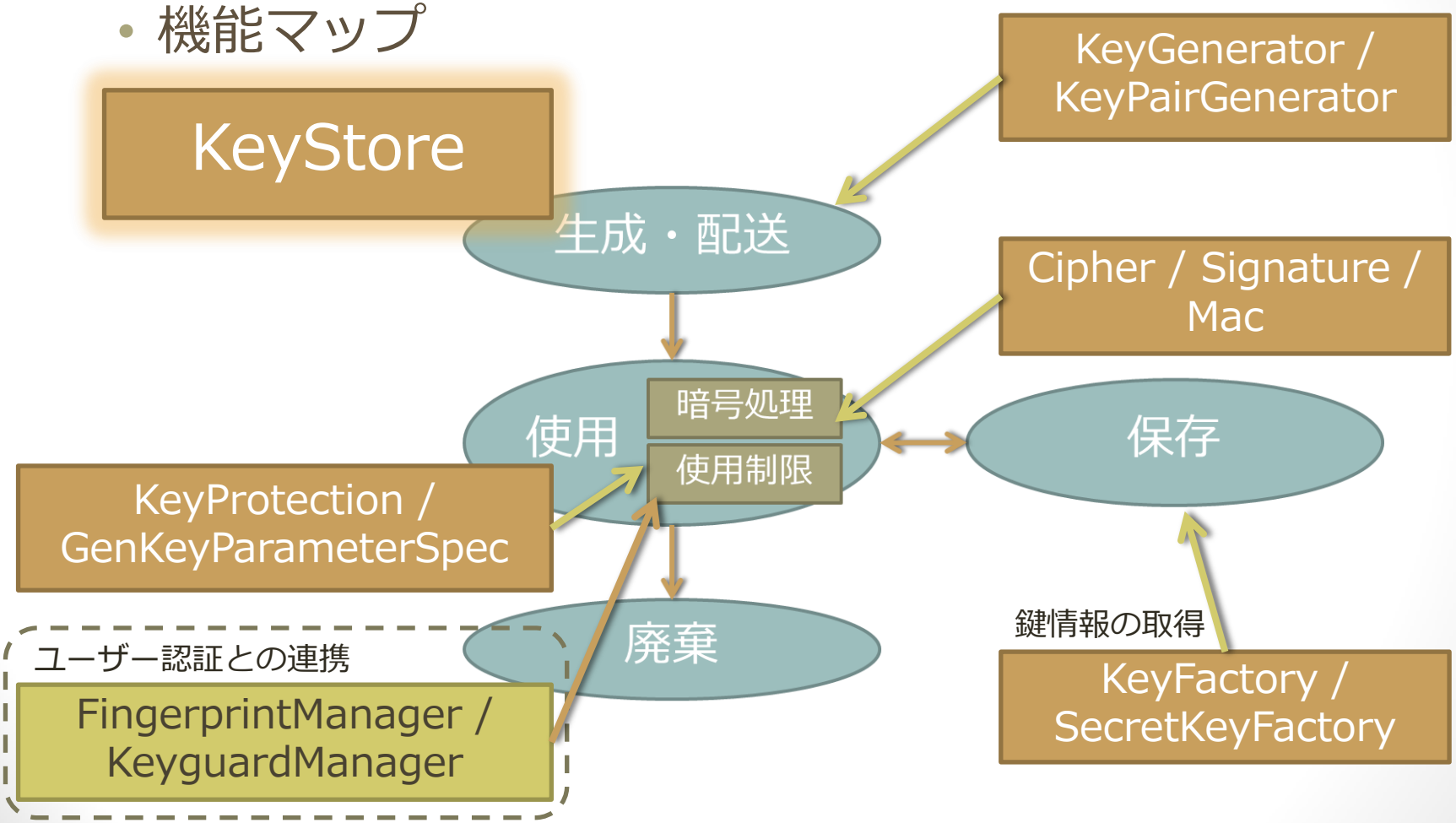
- 鍵の管理＆暗号処理機能を提供するProvider
  - Android 6.0 (API Level 23) にて機能が大きく拡張された
    - 鍵管理機能の拡充
    - 対応アルゴリズムの拡張
      - 一覧 (サイトリンク)
    - ハードウェアによる鍵の保護
    - ユーザー認証




※ API Level 22 以前は、一般アプリから使用可能な機能は、公開 鍵の生成・登録と証明書の取り出しなど

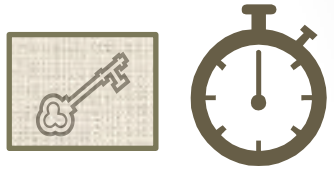
# AndroidKeyStore とは

- 機能マップ

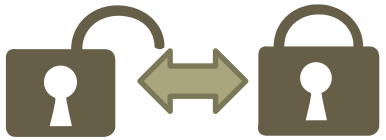


# AndroidKeyStore とは

- 特徴
  - 鍵毎の使用方法/期限/制限
    - KeyProtection / KeyGenParameterSpec

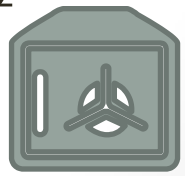


  - 暗号ライブラリとして使用可能
    - 充実したアルゴリズムサポート



  - 鍵使用時のユーザー認証
    - 指紋認証・キーガード(画面ロック)

  - ハードウェアによる保護(hardware-backed)
    - 鍵を TEE など Secure World で扱う
    - 機種(端末毎の実装)や鍵の設定に依存する



# 鍵毎の使用方法/期限/制限 (KeyProtection/KeyGenParameterSpec)

- 下表の鍵の使用目的や保護基準を設定できる

| | | 説明 |
|---|---|---|
| 使用目的 | | 暗号化・復号・署名・検証のどれに使うか |
| ブロック暗号モード | | ECB/CBC/CTR/GCM |
| ダイジェストアルゴリズム | | NONE/MD5/SHA1/SHA224/SHA256/SHA384/SHA512 |
| パディング方式(暗号化、署名) | | 暗号化：NONE/PKCS7, RSA_OAEP/PKCS1, 署名：RSA_PKCS1/PSS |
| 鍵長 | | 鍵の長さ |
| 有効期限 | 有効期限(開始、終了) | 鍵の有効期限 |
| | 暗号化・署名期限(終了) | 暗号化・署名に対する使用期限 |
| | 復号・署名検証期限(終了) | 復号・署名検証に対する使用期限 |
| 証明書パラメータ (KeyGenParameterSpec のみ) | 有効期限(開始、終了) | 鍵(ペア)の証明書の有効期限 |
| | サブジェクト | CN 等、証明書のサブジェクト(Key, Value の対) |
| | シリアルナンバー | シリアルナンバー |
| ランダムネス要求 | | IND-CPA に対する耐性を持った設定を要求する |
| ユーザー認証の有無 | | 指紋認証の有無 |
| ユーザー認証完了後の有効時間 | | キーガード(画面ロック)認証後の有効時間 |

# 暗号ライブラリとして使用可能

- 暗号アルゴリズム (追加分)
  - 暗号化アルゴリズム (Cipher)
    - AES, RSA-OAEP
  - MAC アルゴリズム (Mac)
    - HMAC
  - 署名アルゴリズム (Signature)
    - RSA-PSS
- 鍵の生成 (追加分) (KeyGenerator/KeyPairGenerator)
  - EC, AES, HMAC
- 鍵の情報取得 (KeyFactory/SecretKeyFactory)

アルゴリズムの充実により
暗号ライブラリとして利用可能になった

# 鍵使用時のユーザー認証

- AndroidKeyStore では、鍵の使用をユーザー認証によって制限することができる
- 用意されているシナリオは 2 つ
  - 鍵を使用する毎に毎回認証する
    - 指紋認証によるユーザー認証
      - FingerprintManager

  - 認証後一定時間は鍵の使用を有効にする
    - 画面ロックで設定した認証
      - KeyguardManager

# 指紋認証による鍵の使用制限

- 要件
  - 鍵を使用する度に毎回認証を行いたい
  - 認証に対する操作を簡略化したい
    - 例：楽天銀行アプリがログインに使用

# 指紋認証による鍵の使用制限

- 前提条件
  - 端末に指紋認証機能が付いていること
    - FingerprintManager.isHardwareDetected()
  - 端末に画面ロックが設定されていること
    - KeyguardManager.isKeyguardSecure()
    - 画面ロックが設定されていないと指紋の登録ができない
  - 端末に指紋が登録されていること
    - FingerprintManager.hasEnrolledFingerprints ()

# 指紋認証による鍵の使用制限

- 処理の流れ

1. 鍵の生成・登録

```
try {
    KeyGenerator kg = KeyGenerator.getInstance(KeyProperties.KEY_ALGORITHM_AES,
"AndroidKeyStore");
    kg.init(new KeyGenParameterSpec.Builder(KEY_NAME,
            KeyProperties.PURPOSE_ENCRYPT | KeyProperties.PURPOSE_DECRYPT)
            .setBlockModes(KeyProperties.BLOCK_MODE_CBC)
            .setUserAuthenticationRequired(true)
            .setEncryptionPaddings(KeyProperties.ENCRYPTION_PADDING_PKCS7)
            .build());
    kg.generateKey();
    return true;
} catch (IllegalStateException e) {
        // This happens when no fingerprints are registered.
} catch (NoSuchAlgorithmException | InvalidAlgorithmParameterException
        | CertificateException | IOException e) {
}
```

指紋認証の要求

指紋が登録されていないと例外が発生

# 指紋認証による鍵の使用制限

- 処理の流れ

2. 暗号化処理の準備

```
try {
        KeyStore ks = KeyStore.getInstance("AndroidKeyStore");
        ks.load(null);
        SecretKey key = (SecretKey) ks.getKey(KEY_NAME, null);
        mCipher = Cipher.getInstance(KeyProperties.KEY_ALGORITHM_AES + "/"
                                     + KeyProperties.BLOCK_MODE_CBC + "/"
                                     + KeyProperties.ENCRYPTION_PADDING_PKCS7);
        mCipher.init(Cipher.ENCRYPT_MODE, key);
        return true;
} catch (KeyPermanentlyInvalidatedException e) {
} catch (KeyStoreException | CertificateException | UnrecoverableKeyException |
IOException | NoSuchAlgorithmException | InvalidKeyException e) {
}
```

登録時の制限に合ったアルゴリズムを選択

制限に合わないと例外が発生

# 指紋認証による鍵の使用制限

- 処理の流れ

3. 認証の開始

セッションとして働く

```
CtyptoObject cryptoObject = new FingerprintManager.CryptoObject(mCipher);
mCancellationSignal = new CancellationSignal();
mFingerprintManager.authenticate(cryptoObject, mCancellationSignal, 0 /* flags */,
this /*callback*/ , null);
```

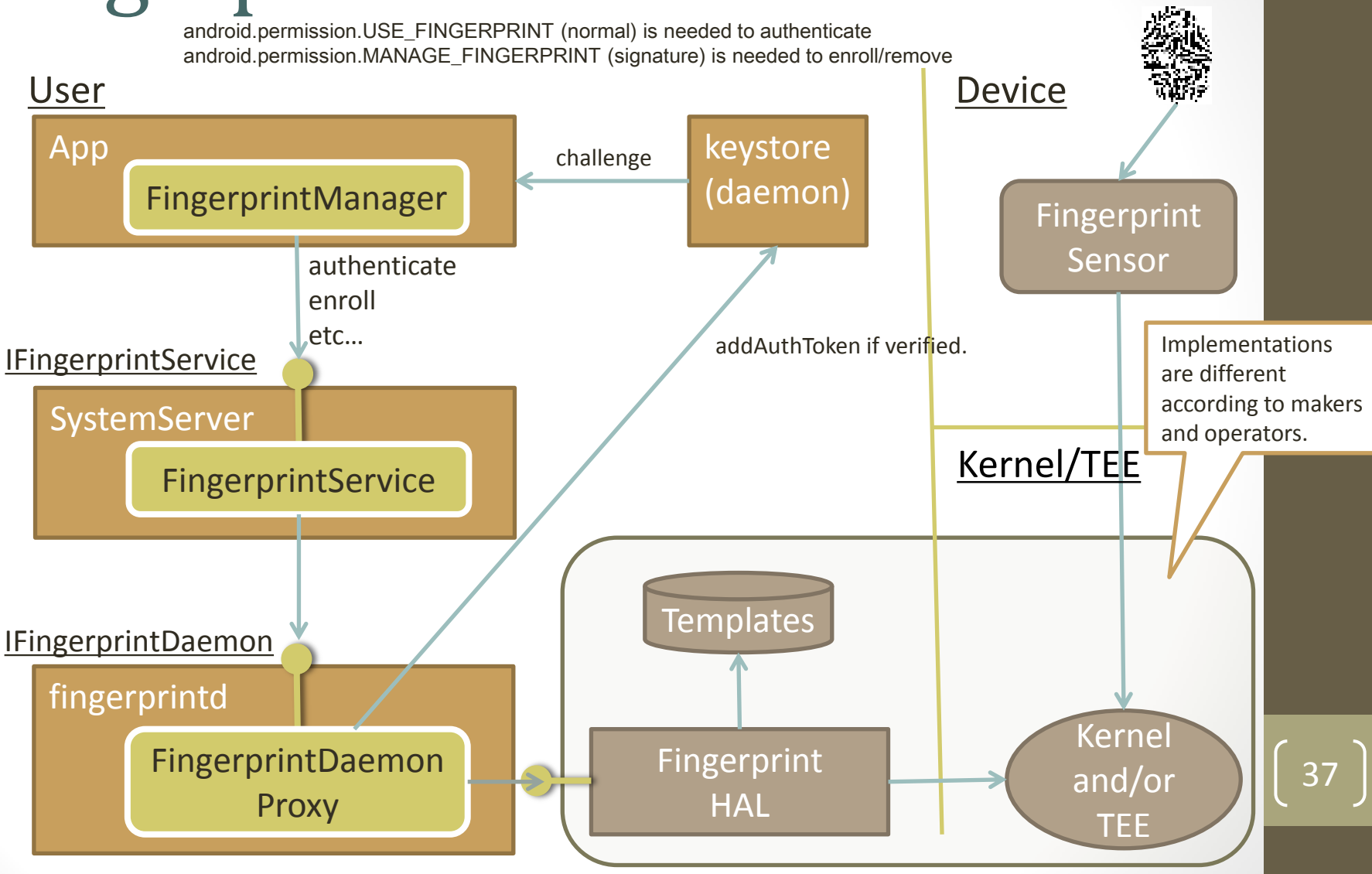
認証開始

4. 暗号処理 (Callback に成功・失敗が返る)

```
public void onAuthenticationSucceeded(FingerprintManager.AuthenticationResult result)
{
   try {
      Cipher chiper = result.getCryptoObject().getCipher();
      byte[] encrypted = cipher.doFinal(SECRET_MESSAGE.getBytes());
      doSomothing(encrypted);
   } catch (BadPaddingException | IllegalBlockSizeException e) { }
}
```

認証成功はコールバックで処理

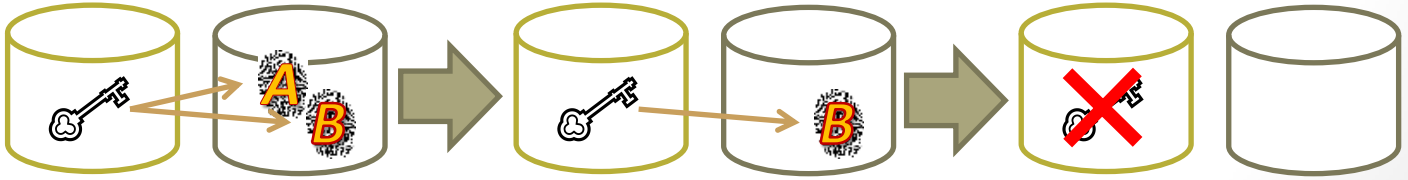
# Fingerprint Architecture



# 指紋認証による鍵の使用制限

- 注意事項①
  - 鍵の無効化
    - 生成・登録時に設定した条件以外に鍵が無効化になるケース(下記)があるので注意すること
      - 鍵生成・登録後に、端末に新たに指紋が登録された場合


      - 鍵生成・登録時に登録されていた指紋がすべて消去された場合



# (参考)指紋認証の特徴 

- パスワード (PIN, Pattern) に比べて
  - Pros
    - 入力が容易である
    - 覚える必要がない
    - 一定の強度がある
      - 弱いパスワードよりは強い
  - Cons
    - 認証としての(絶対的)強度は高くない
      - 認証精度、秘匿困難性、人工物による認証
    - 指紋が漏れても変えられない
      - 非交換性
    - 環境による識別精度の低下（誤認識）が存在する
      - 怪我、経年変化、外的要因(湿度、明度、気温 etc…)
      - パスワードのように一意に決まらない

# 指紋認証による鍵の使用制限

- 注意事項②
  - 指紋認証の弱い点を考慮する
    - 指紋認証だけに頼る設計にはしない
      - 認証ができなくなった（生体が変化するなど）でも、別手段で機能を利用できるようにする
    - 機密性の高い情報・機能は（直接）扱わない
      - 課金や決済といった重要な情報・機能を扱う際はかならず他の認証方法と併用する

# 画面ロックによる鍵の使用制限

- 要件
  - 認証後、一定時間は鍵の使用を有効にしておきたい
    - 短時間に複数回使用する場合に認証を省略したい

# 画面ロックによる鍵の使用制限

- 前提条件
  - 端末に画面ロックが設定されていること
    - KeyguardManager.isKeyguardSecure()

# 画面ロックによる鍵の使用制限

- 処理の流れ

1. 鍵の生成・登録

```
try {
    KeyGenerator kg = KeyGenerator.getInstance(KeyProperties.KEY_ALGORITHM_AES,
"AndroidKeyStore");
    kg.init(new KeyGenParameterSpec.Builder(KEY_NAME,
            KeyProperties.PURPOSE_ENCRYPT | KeyProperties.PURPOSE_DECRYPT)
            .setBlockModes(KeyProperties.BLOCK_MODE_CBC)
            .setUserAuthenticationRequired(true)
            .setUserAuthenticationValidityDurationSeconds(AUTHENTICATION_DURATION_SECONDS)
            .setEncryptionPaddings(KeyProperties.ENCRYPTION_PADDING_PKCS7)
            .build());
     kg.generateKey();
     return true;
} catch (NoSuchAlgorithmException | NoSuchProviderException
        | InvalidAlgorithmParameterException | KeyStoreException
        | CertificateException | IOException e) {
}
```

ユーザー認証の要求に加えて、有効時間を指定する

# 画面ロックによる鍵の使用制限

- 処理の流れ

2. 暗号化の実行：認証エラー時は認証画面表示

```
private void tryEncrypt() {
   try {
      KeyStore keyStore = KeyStore.getInstance(“AndroidKeyStore”);
      keyStore.load(null);
      SecretKey secretKey = (SecretKey) keyStore.getKey(KEY_NAME, null);
      Cipher cipher = Cipher.getInstance(KeyProperties.KEY_ALGORITHM_AES + “/” +
KeyProperties.BLOCK_MODE_CBC + “/” + KeyProperties.ENCRYPTION_PADDING_PKCS7);
      cipher.init(Cipher.ENCRYPT_MODE, secretKey);
      cipher.doFinal(SECRET_BYTE_ARRAY);
   } catch (UserNotAuthenticatedException e) {
     showAuthenticationScreen();
   } catch (KeyPermanentlyInvalidatedException e) {
   } catch (…/*省略*/) {   }
}
```

鍵の生成時に設定した認証期限が切れていると例外発生するので、認証画面を呼び出す

# 画面ロックによる鍵の使用制限

- 処理の流れ

3. 認証処理の開始(認証[ロック]画面の表示)

```
private void showAuthenticationScreen() {
   Intent intent = mKeyguardManager
                   .createConfirmDeviceCredentialIntent(“title", “description");
   if (intent != null) {
      startActivityForResult(intent, REQUEST_CODE_CONFIRM_DEVICE_CREDENTIALS);
   }
}
```

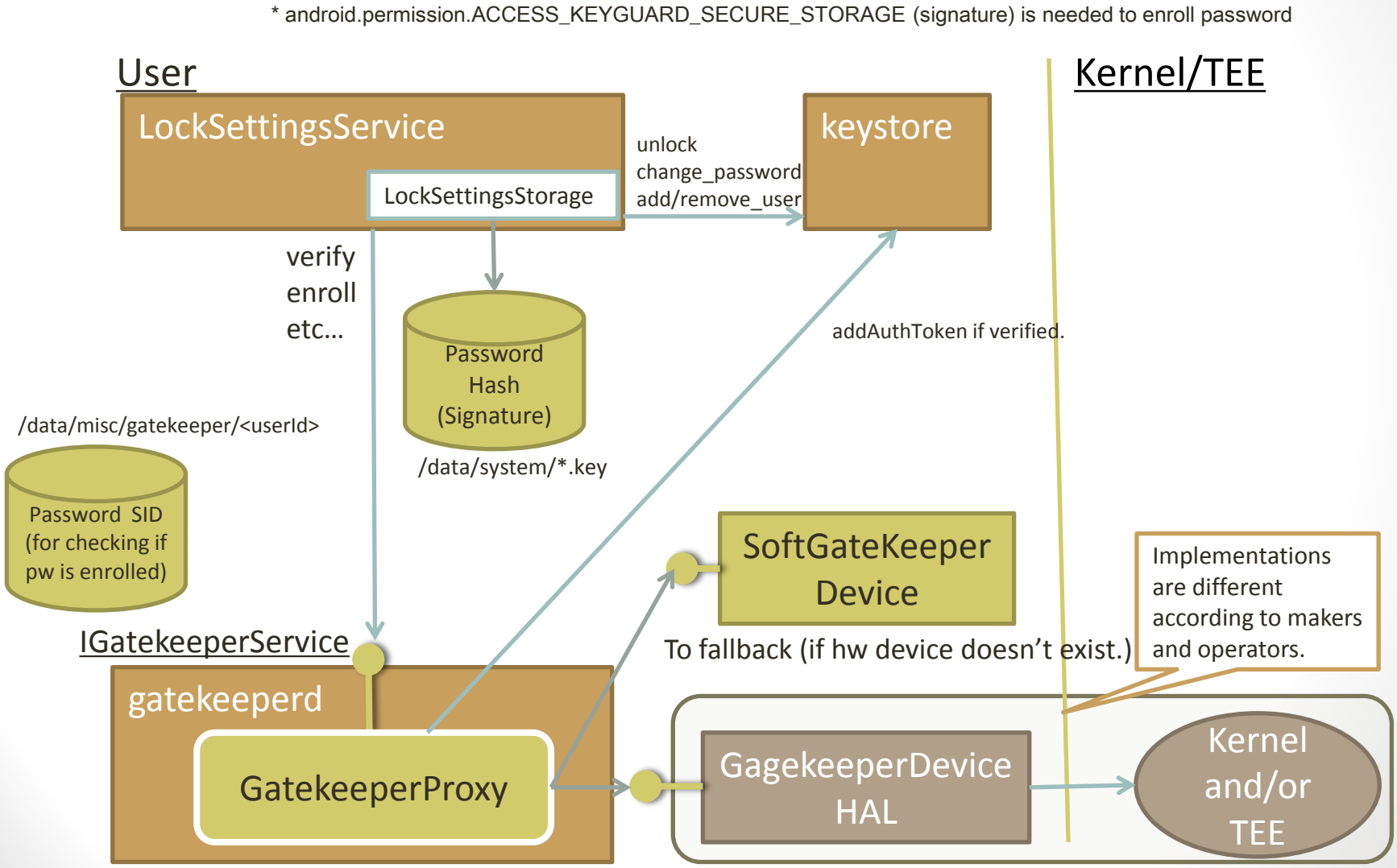
startActivityForResult で認証開始

4. 暗号処理の実行 (認証成功時)

```
protected void onActivityResult(int requestCode,  int resultCode, Intent data) {
   if (requestCode == REQUEST_CODE_CONFIRM_DEVICE_CREDENTIALS &&
         resultCode == RESULT_OK) {
      tryEnctyption();
   }
}
```

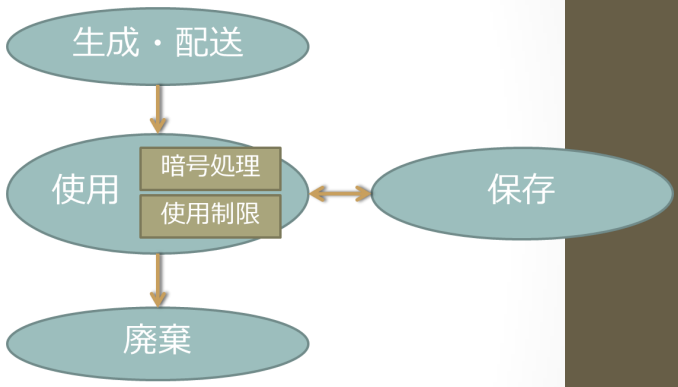
認証 OK の場合、改めて暗号処理を行う
※鍵の生成時に設定した時間は鍵の使用が可能

# Gatekeeper Architecture

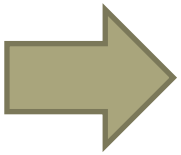


# AndroidKeyStoreにおける鍵のハードウェア保護

- ハードウェア保護とは
  - 鍵のライフサイクルを TEE など Secure World で処理すること
    - 生成から使用・保存・廃棄まで
  - 現状では最も安全な守り方の一つ




- 可用性は？
  - 機種(端末毎の実装)や鍵の設定に依存する
    - 機種、アルゴリズム、鍵の設定 etc...



鍵が HW 保護されているかどうかは鍵毎に確認が必要

# AndroidKeyStoreにおける鍵のハードウェア保護

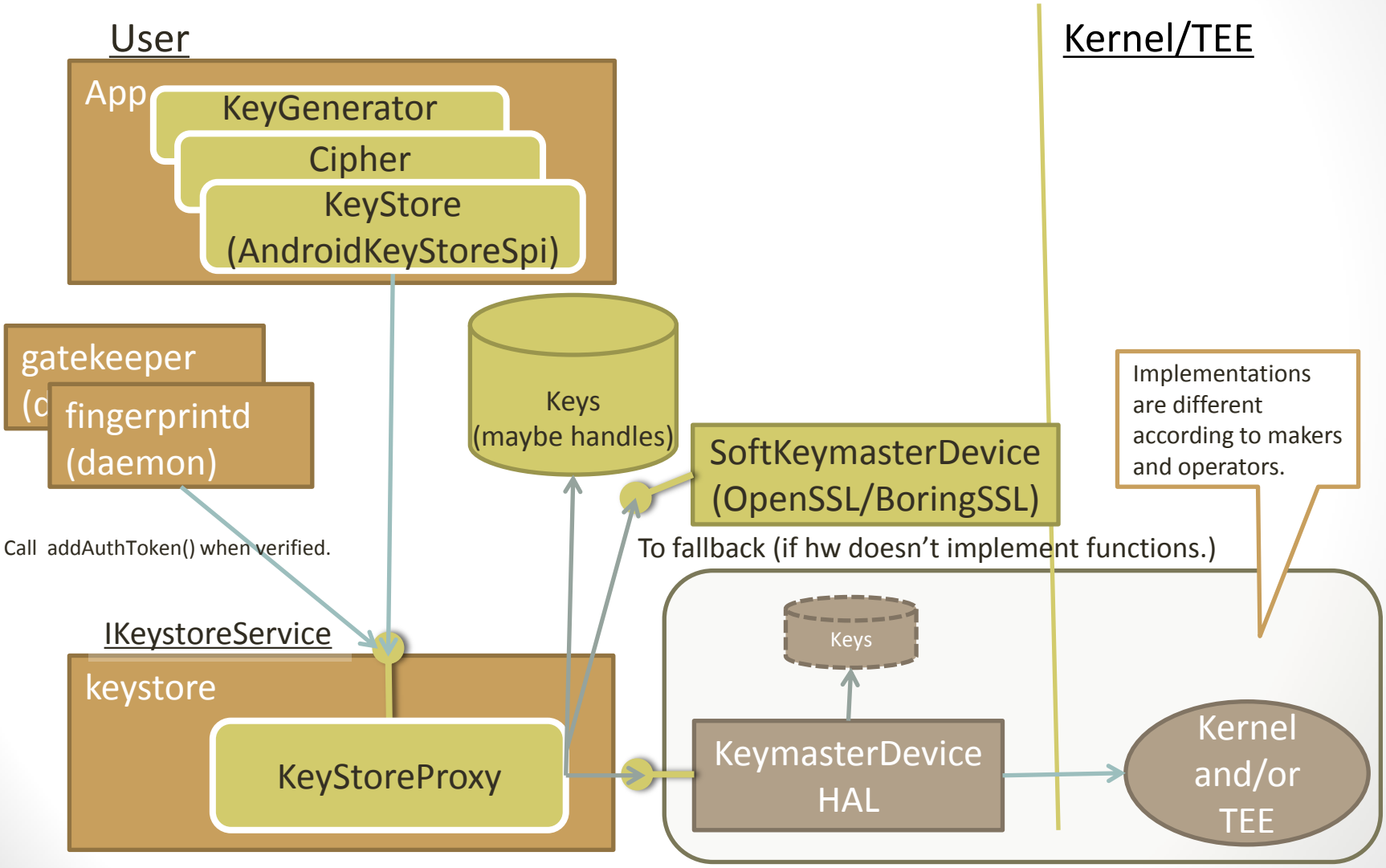
- 鍵のハードウェア保護の確認方法
  - 以下、サンプルコード

```
try {
   KeyStore ks = KeyStore.getInstance("AndroidKeyStore");
   ks.load(null);
   SecretKey secretKey = (SecretKey) ks.getKey(KEY_NAME, null);
   SecretKeyFactory kf = SecretKeyFactory.getInstance(KeyProperties.KEY_ALGORITHM_AES,
"AndroidKeyStore");
   KeyInfo ki = (KeyInfo) kf.getKeySpec(key, KeyInfo.class);
   if (ki.isInsideSecureHardware() &&
        ki.isUserAuthenticationRequirementEnforcedBySecureHardware())
      return true;
} catch (…/*省略*/) {
}
```

鍵が HW によって守られているか

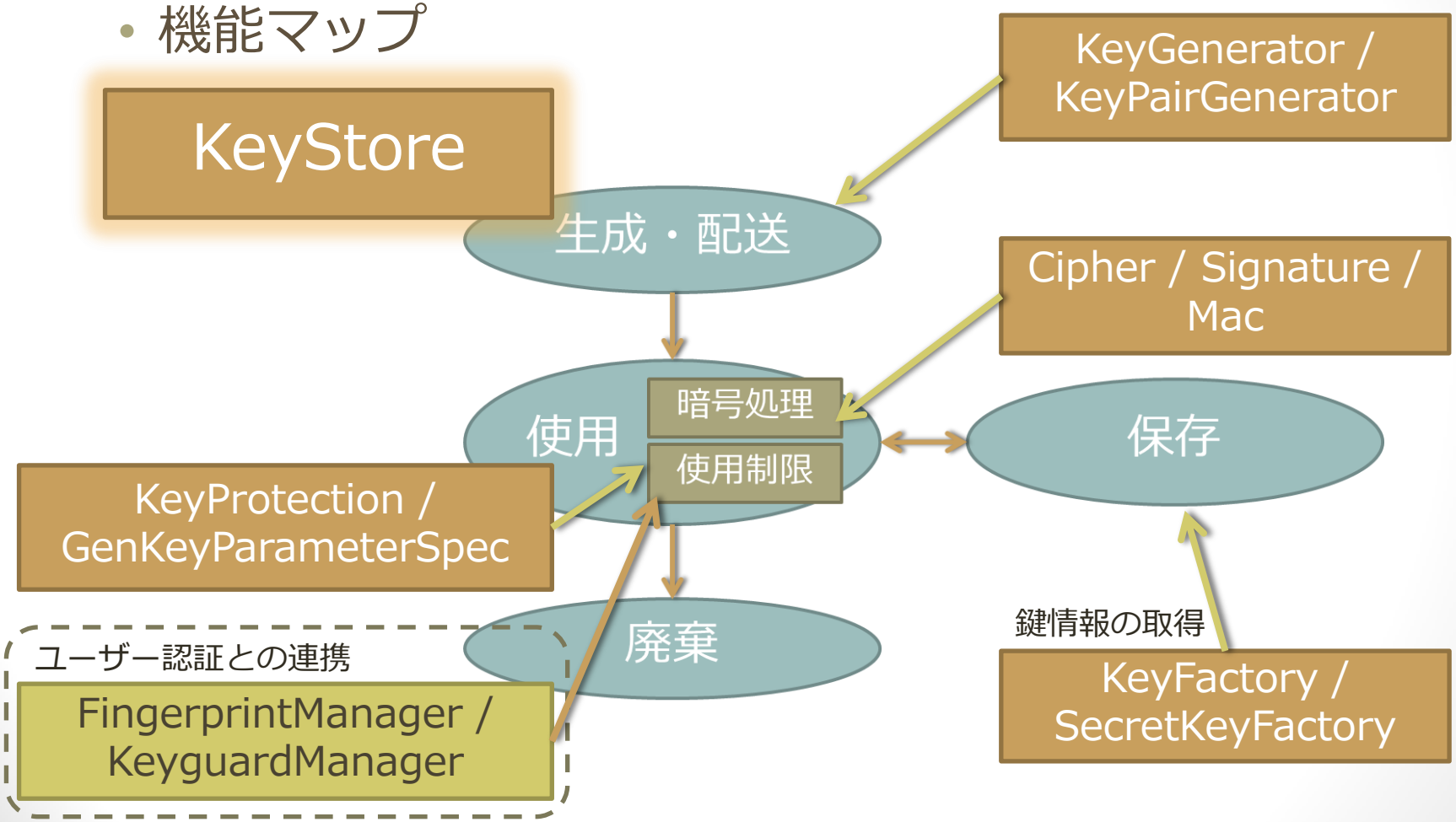
認証処理が HW によって守られているか

# AndroidKeyStore Architecture



# AndroidKeyStore とは

- 機能マップ



Android N Preview 段階における

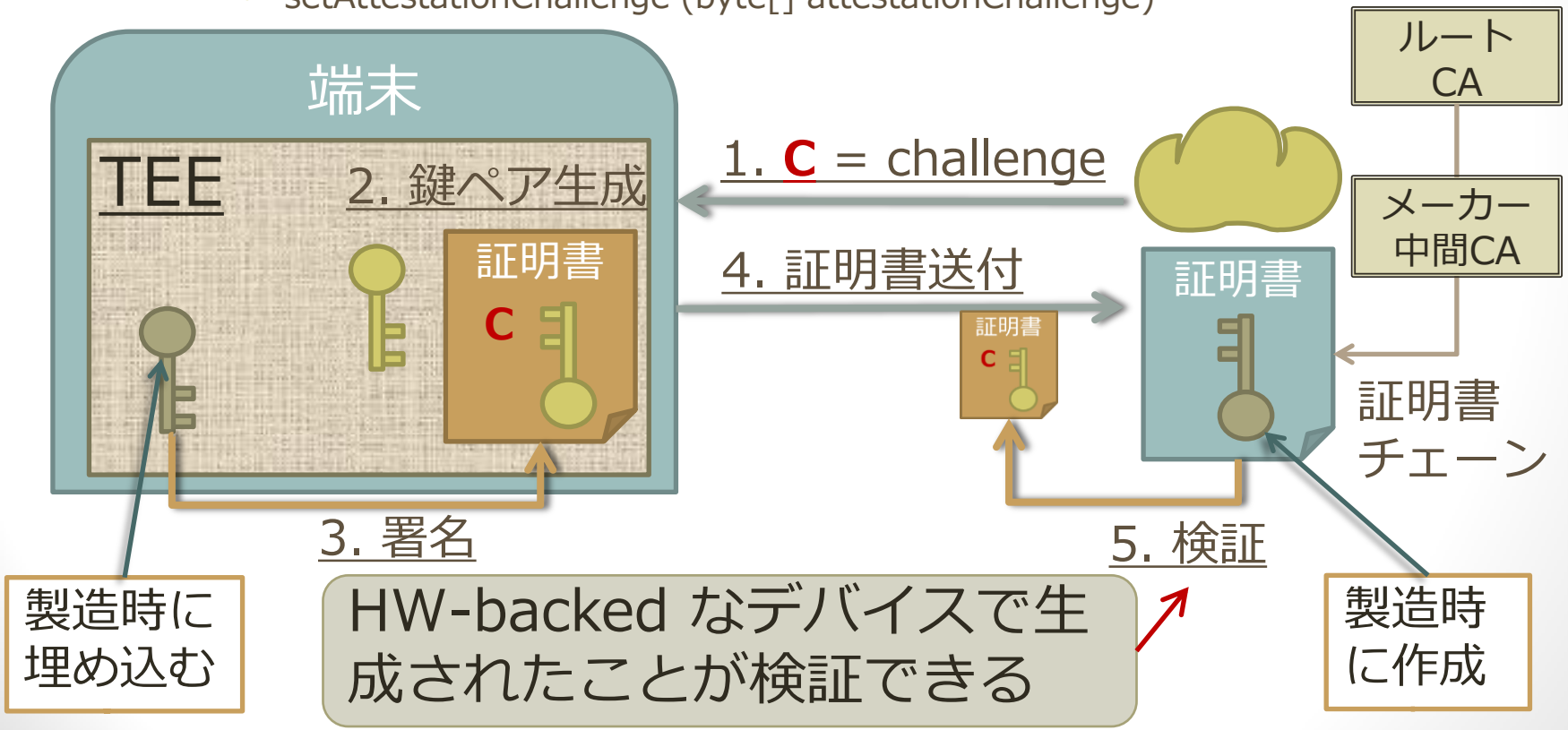
# ANDROID N の追加機能

# Android N の追加機能

- AndroidKeyStore に関わる機能が追加されている
  - Key Attestation
    - 生成した鍵にHW-backedで第三者検証可能な証明書を発行する
  - On-body detection
    - ユーザーが端末を携帯している間は鍵の使用を有効に保つ
  - 鍵の有効性の制御(指紋認証)
    - 指紋登録の追加・(全)削除に対して、鍵の有効性を継続する手段が提供される

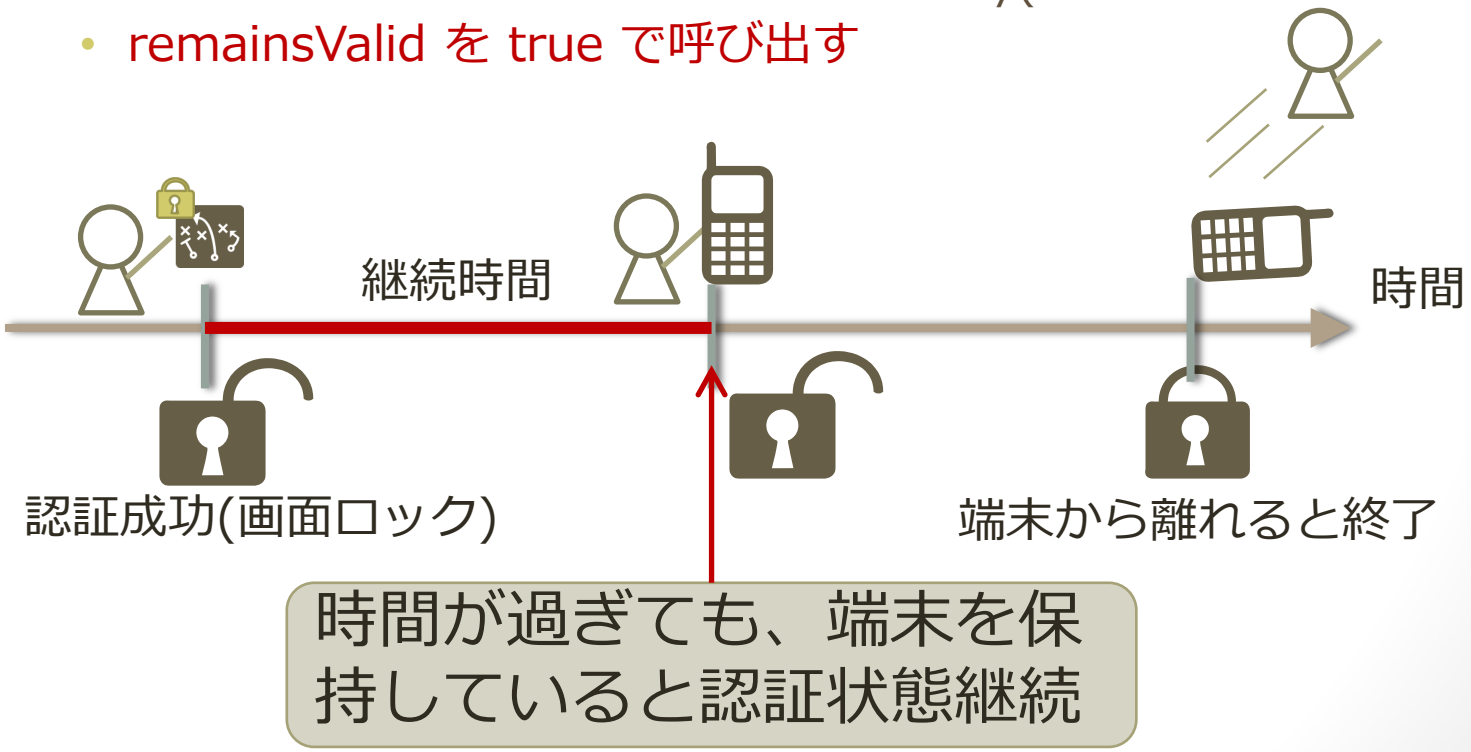
# Android N の追加機能

- Key Attestation
  - 生成した鍵にHW-backedで第三者検証可能な証明書を発行する
  - KeyGetParameterSpec クラス
    - setAttestationChallenge (byte[] attestationChallenge)



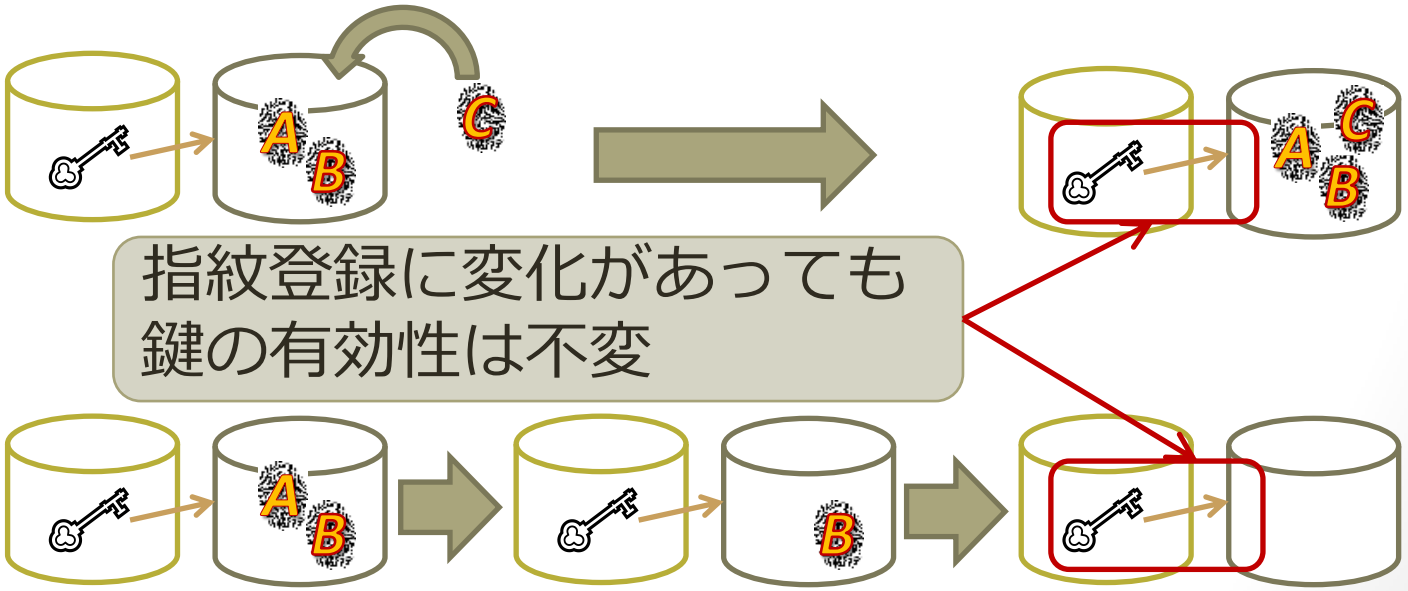
# Android N の追加機能

- On-body detection
  - ユーザーが端末を携帯している間は鍵の使用を有効に保つ
  - KeyProtection / KeyGenParameterSpec クラス
    - setUserAuthenticationValidWhileOnBody(boolean remainsValid)
    - remainsValid を true で呼び出す



# Android N の追加機能

- 鍵の有効性の制御(指紋認証)
  - 指紋登録の追加・(全)削除に対して、鍵の有効性を継続する手段が提供される
  - KeyProtection / KeyGenParameterSpec クラス
    - setInvalidatedByBiometricEnrollment(boolean invalidateKey)
    - invalidateKey を false で呼び出す



# おわりに

- Android アプリにおける鍵管理
- Android 6.0 の AndroidKeyStore
  - 鍵のライフサイクルをカバーしている
  - 鍵の管理機能も装備されている
  - Android N でも拡張が進みそう

  - 機種によってHW保護の状況が異なる

- 今後 Android アプリでの鍵管理や暗号処理を AndroidKeyStore で行うことが一般的になるかも知れません