ハードウェアベース暗号鍵管理に関する日本向けAndroidプラットフォームの調査

磯部 光平[1,a)]　坂本 一仁[1]　葛野 弘樹[1]

概要：Android や iOS のようなスマートデバイス向けオペレーティングシステム（OS）では，ハードウェアによる暗号鍵管理機能（鍵管理機能）が提供されている．ユーザのクレデンシャルがハードウェアによる鍵管理機能によって強固に保護されている場合，高いセキュリティが実現されていると言える．一方，スマートデバイスの製造業者が鍵管理機能を利用可能な状態でデバイスを提供し，さらにアプリケーション（アプリ）が鍵管理機能を利用しているかどうかは不透明な状況にある．本稿では，Android OS を搭載したスマートデバイスに着目し，暗号鍵管理の使用実装状況を調査した．アプリではライブラリを中心に広く鍵管理機能が利用されていることを確認した．デバイスでは調査した 71 機種のすべての機種で RSA はハードウェアによる鍵管理機能が利用可能なものの，楕円曲線暗号や HMAC において，利用不可な機種が一定数存在することを確認した．

An Investigation for Japanese Android Platform related to Hardware-backed Key Management.

Kohei Isobe[1,a)]　Takahito Sakamoto[1]　Hiroki Kuzuno[1]

Abstract: Smart device's OSs (e.g., Android, iOS) provide a hardware-backed key management functions. Under the circumstances that users' credentials are protected by hardware-backed key management, the smart devices of the users are protected with higher security environment. Meanwhile, it is not clear that smart devices on the market support hardware-backed key management and applications utilize key management function. In this paper, we investigated the adoptions and utilizing rates of key management function in Android platform. We found that many applications used various key management functions by loading libraries. All 71 devices that we investigated supported RSA functions by hardware-backed key management. However, some devices did not support ECDSA and HMAC functions.

1. はじめに

Internet of Things（IoT）の拡大とともに，ネットワーク経由で機器の遠隔制御や状況の確認が可能な製品が数多く登場している．これらの製品は，ユーザの操作用にスマートフォンやタブレットなどのスマートデバイス向けの専用アプリケーション（以降，アプリ.）が提供されている．ユーザは専用アプリを自身のスマートデバイスにインストールし，製品とのペアリングを行ったうえで製品を利用する．このとき，専用アプリには第三者からの製品に対する不正な操作を防ぐための認証情報が保管されることが多い．一方で，スマートデバイスにはルート化や脱獄と呼ばれるソフトウェアに対する改造行為が存在する．スマートデバイス内のソフトウェアに対して改造が行われるとデバイス上のアプリに保存された情報の漏洩やセキュリティリスクが生じる．このようなリスクに対して，ソフトウェアのみに依存しないハードウェアベースのクレデンシャル管理が有効である．iOS や Android といったスマートデバイス向け OS はクレデンシャル管理に利用可能な機能として暗号鍵管理機能を備えている．実際にハードウェアによる保護を有効とするには OS とハードウェアとの統合が不可欠なため，スマートデバイスメーカによる実装が行われ

[1] セコム株式会社 IS 研究所
Intelligent Systems Laboratory, SECOM CO., LTD.
[a)] ko-isobe@secom.co.jp

るべきである．特に Android プラットフォームではデバイスメーカが多く存在し，個々のメーカや機種における実装状況が不透明であり，アプリで利用するクレデンシャルがハードウェアによる保護を受けているかは定かではない．また，アプリにおいても実際にどのくらいのアプリが鍵管理機能を利用しているのか，不透明な状況にある．

そこで本稿では，日本向け Android プラットフォームに対する 2 つの調査を実施した．1 つ目の調査として，日本国内で販売・流通している Android OS 搭載端末のハードウェアベース暗号鍵管理機能（鍵管理機能）の対応状況を調べた．もう 1 つの調査として，日本向け Android アプリにおける鍵管理機能の利用状況を調査した．その結果，スマートデバイスにおいては，調査対象とした 71 機種のうち，すべての機種で RSA 暗号は鍵管理機能に対応していることを確認した．一方で，楕円曲線暗号や HMAC に関しては，鍵管理機能に非対応の機種が存在した．また，17,757 の日本向けアプリを調査した結果，76.9%のアプリで鍵管理機能 API の実装が確認されたものの，そのほとんどはライブラリでの利用であり，アプリのメイン機能での利用は少数であった．

以降，2 章ではスマートデバイスの構成と暗号鍵管理手法について，3 章では本稿で行った調査手法について述べ，4 章では調査結果を示す．5 章で調査結果に対する考察を与え，6 章でまとめる．

2. スマートデバイスのアーキテクチャと鍵管理

2.1 アーキテクチャ

スマートデバイスは，CPU や GPU を搭載したモバイル向け SoC，タッチパネルやセンサ等の各種周辺機器，スマートデバイス用 OS，サードパーティアプリで構成される．Apple 社のスマートデバイスである iPhone や iPad は，ハードウェアと OS を同社が一貫して開発し，管理している．Android OS 搭載デバイスは，開発元の細かな差異はあるものの多くのハードウェアは共通化されている．

デバイス上で動作させる OS やサードパーティアプリを共用するためには，CPU アーキテクチャなどを同一としなければならない．そのため，一般的には主要なハードウェアである CPU アーキテクチャに ARM を採用しており，Qualcommm 社などの ARM 社からライセンスを受けた企業の開発した SoC を搭載し，周辺機器にはデバイスドライバの存在するハードウェアを配置し，OS を動作させることでスマートデバイスを構成している．

Android OS において，多種多様なハードウェアに対応するために Hardware Abstraction Layer（HAL）を導入しており，スマートデバイスごとに生じるハードウェアの違いや OS のバージョンにより提供可能なハードウェア機能の差異を OS にて可能な限り低減させることを試みている．

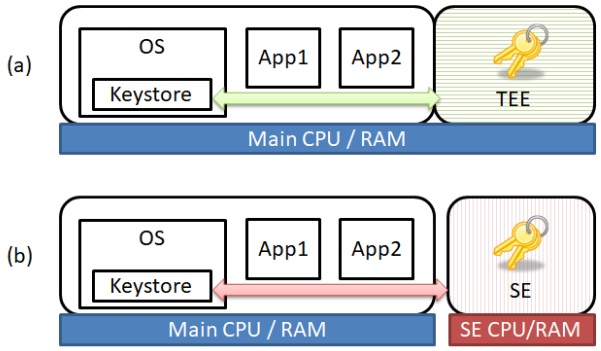

図 1　スマートデバイスにおける鍵管理の構成パターン
Fig. 1　Composition patterns of key management in smart devices.

実際，Android OS のバージョンにより，提供されるライブラリや API に違いは存在するものの，サードパーティアプリが多数のスマートデバイスで動作することを可能としている．

サードパーティアプリの開発者は，通常はスマートデバイス向け OS の開発元が提供するプラットフォーム（API や SDK）を利用する．クレデンシャル保護に利用できる暗号鍵管理に関しても，プラットフォーム側で API が提供されている．Android OS の場合は，Keystore[1] であり，iOS は Keychain[2] である．これら API を使うことで，アプリ開発者はスマートデバイス毎のハードウェアの差異を意識せず，暗号鍵管理を行うことができる．Keystore API は 2013 年にリリースされた Android OS 4.3 より利用可能となった [3]．サードパーティアプリ開発者は，Keystore API を利用することで，個々のアプリ単位での暗号鍵の生成，利用，破棄が可能となる．

2.2 暗号鍵管理の構成

ハードウェアベースの暗号鍵管理を構成する上で，Keystore は大きく分けて 2 つの構成をサポートする．その構成を図 1 に示す．

Keystore がリリースされた当初は，OS 内のソフトウェア処理での暗号鍵管理をサポートしていた．したがって通常アプリと同一のメモリ空間で暗号鍵に関する処理が行われるため，OS のアクセスコントロール機構が OS の改造等によって無効化された場合，暗号鍵の危殆化に繋がるリスクがあった [4]．

第一に TEE（Trusted Execution Environment）を用いた構成がある (図 1（a）)．TEE はメモリ空間をハードウェアによって分離することで複数のアプリケーションの実行領域を独立して確保する手法である．暗号鍵管理を OS やサードパーティアプリが動作する実行領域と独立した領域で実行することで OS へ介入されてもその影響が暗号鍵管理に及ばないようにする．TEE のハードウェア実装として ARM 社の TrustZone[5] がある．Qualcomm 社などは

自社の SoC に TrustZone のハードウェア実装を取り込み，TEE を利用可能としている．2015 年リリースの Android OS 6.0 では，暗号鍵管理を TEE で分離された空間で取り扱うことに対応した [6]．また，2016 年にリリースされた Android OS 7.0 で，Google 社は TEE を含めたハードウェアベースの鍵管理機能の実装を必須とした [7].

第二の構成は，暗号処理専用のチップを設ける構成である (図 1（b）)．これは，OS やアプリを実行するメインの CPU・メモリとは別に暗号処理専用に CPU やメモリを設け，物理的に処理を独立させる形態である．物理的に異なる箇所で処理されるため，メイン CPU 上で実行される OS やアプリへの介入の影響を受けることがない．暗号処理専用の実装は Secure Element（SE）とも呼ばれ，実装例として Google 社の Titan M チップ [8] や Qualcomm 社の SoC である Snapdragon 845 に搭載された Secure Processing Unit[9] がある．SIM カードを SE として用いる構成もある．Android OS では，バージョン 8.0 で StrongBox という名称で専用チップの HAL での定義に対応した [1].

これら API を使うことで，アプリ開発者は各デバイスのハードウェア上の差異を意識せず，ハードウェアベースの暗号鍵管理を利用することが可能となっている．一方で，構成を選択し，HAL の構築や OS のビルドなど鍵管理機能を利用可能とするように実装を行うのはスマートデバイスメーカの役割である．そのため，各デバイスメーカやデバイス毎にハードウェアベース暗号鍵管理の対応状況に差異があることが想定される．以上から市場で流通する種々のスマートデバイスのハードウェアベース鍵管理の対応状況は不透明にある．

3. 調査

2 章では Android OS を採用したスマートデバイスでは，ハードウェアベース暗号鍵管理機能の対応に差異が存在しうることを述べた．本稿では，Android プラットフォーム上での暗号鍵管理の実装状況の把握を目的とし，次の 2 種類の調査を実施した．

(1) デバイス：ハードウェアベース暗号鍵管理機能の実装状況
(2) アプリ：暗号鍵管理機能の利用状況

3.1 ハードウェアベース鍵管理の実装状況

Android デバイス側に対する調査は、調査対象を日本国内で近年入手可能な Android 搭載デバイスとした．具体的にはハードウェア対応が必須化された Android OS 7.0 以降がインストールされたデバイス 71 機種を準備した．71 機種には，同一ハードウェアの仕向け違い (主に販売元となる携帯電話キャリア) や OS バージョン違いが含まれるが，これらは別機種としてカウントしている．調査対象デバイスの OS バージョン，発表時期，デバイスメーカを表 1,2,3 にそれぞれ示す.

表 1　対象機種数 (OS 別)
Table 1　Numbers of target devices (by OSs).

バージョン	機種数
7 系	3
8 系	15
9 系	53
合計	71

表 2　対象機種数 (発売時期別)
Table 2　Number of target devices (by released year).

発表年	機種数
2015	1
2016	3
2017	18
2018	30
2019	19

表 3　テスト対象の機種数 (メーカ別)
Table 3　Number of target devices (by device makers).

メーカ名	機種数	メーカ名	機種数
Samsung	20	Huawei	3
Sharp	19	Kyocera	2
Sony	18	asus	1
Google	3	Panasonic	1
Fujitsu	3	ZTE	1

表 4　テスト対象の暗号アルゴリズム
Table 4　List of target cryptographic algorithms.

アルゴリズム	パターン
RSA	鍵長 (bit)：2048, 3072, 4096
ECDSA	楕円曲線： P-224, P-256, P-384, P-521
AES	鍵長 (bit)： 128, 256
HMAC	ハッシュアルゴリズム： SHA1, SHA224, SHA256, SHA384, SHA512

対応状況の確認には、Android OS が提供する API を使用した。Android OS 6.0 では，Keystore を用いて生成した暗号鍵がハードウェアベースで管理されているかどうかを確認するメソッド `KeyInfo#isInsideSecureHardware()` が追加された [10]．スマートデバイス上で生成した鍵に対してこの API を用いた際に true が出力された場合，当該スマートデバイスではその暗号鍵を TEE や SE を用いて管理していると確認できる．本調査では，Keystore が対応する複数の暗号アルゴリズムに対して鍵生成を実施し，API の出力を集計した．鍵生成と API による確認をリスト 1 に示す．Keystore が対応する暗号アルゴリズムを表 4 に示す.

上述の確認手法を実装した調査用アプリを開発した．アプリには確認機能に加え，端末の OS バージョンなど本体

リスト 1　鍵ペアの生成と鍵保存先を確認するコード (RSA2048 ビットの例)

```
//Generate RSA-2048bit KeyPair
KeyPairGenerator kpg = KeyPairGenerator.getInstance(KeyProperties.KEY_ALGORITHM_RSA, "
    AndroidKeyStore");
AlgorithmParameterSpec spec = new KeyGenParameterSpecBuilder.("RSA-2048bit").
    setAlgorithmParameterSpec(new RSAKeyGenParameterSpec(2048, RSAKeyGenParameterSpec.F4)).build()
    ;
kpg.initialize(spec);
KeyPair keyPair = kpg.generateKeyPair();

//Confirm PrivateKey Store
PrivateKey sKey = keyPair.getPrivate();
KeyFactory kFac = KeyFactory.getInstance(KeyProperties.KEY_ALGORITHM_RSA,"AndroidKeyStore");
KeyInfo kInfo = kFac.getKeySpec(sKey,KeyInfo.class);
boolean isStoredHardware = kInfo.isInsideSecureHardware();
```

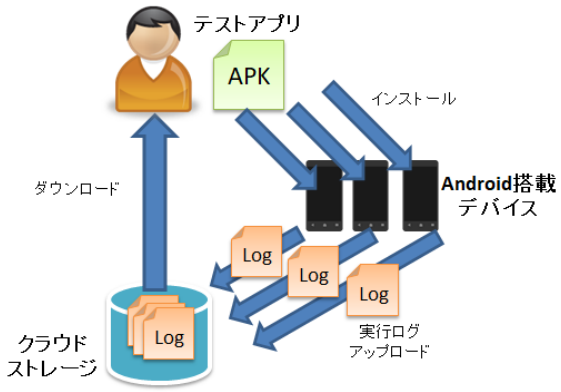

図 2　Android デバイスの調査方法
Fig. 2　Scheme of Testing Android devices.

情報を収集する機能とクラウドストレージにテスト結果や収集した情報をログファイルとしてアップロードする機能を追加実装した．準備した各デバイスに対し，調査用アプリをインストールしアプリを実行する．クラウドストレージにアップロードされたログの集計・分析を行うことで結果を得た (図 2)．

3.2　Android アプリの鍵管理利用状況

アプリ側の利用状況調査では日本国内を対象に配信される Android アプリを調査対象とした．具体的には，Google Play の配信カテゴリ毎に，配信ランキング上位の最大 600 アプリを取得し，33 カテゴリ合計 17,757 アプリを準備した．今回調査対象としたアプリ群は文献 [11] と同一である．

Keystore API の利用状況を確認するために，文献 [11] で実施した静的解析手法を適用した．Keystore API を利用して暗号鍵をアプリ内で生成する場合，鍵生成クラスのインスタンスを生成しなければならない．本調査ではインスタンス生成に着目し，表 5 に示す条件を設定して静的解析を実施した．静的解析では，各アプリにおいて表 5 の $\{p1, p2\}$ を $(p1 \mid p2)$ として抽出した．また，Android アプリのメイン機能に関するソースコードは一般的にパッケージ名に沿ったディレクトリ構造下に配置されている．例えば，アプリのパッケージ名が “com.example.app” であれば，メイン機能は “com/example/app/MainActivity.java” に格納されている．本調査においても文献 [11] のアルゴリズム 1 を踏襲し，アプリのメイン機能側で Keystore API が利用されていれば `main`，サードパーティライブラリ等で利用されていれば `other` と分類し，集計した．

4.　調査結果

4.1　ハードウェアベース鍵管理の実装状況

暗号アルゴリズム別の対応機種数を表 6 に示す．今回テスト対象とした 71 機種はすべて鍵長を問わず RSA のハードウェアベース暗号鍵管理に対応していた．楕円曲線，AES は 1 機種が非対応だった．また特定の鍵長のみの限定的な対応をする機種はなかった．HMAC では 1 機種が HMAC-SHA256 に非対応であった．他のハッシュアルゴリズムに関しては，HMAC-SHA1 が 6 機種で非対応．HMAC-SHA224, 384, 512 は 11 機種で非対応であった．発売時期別に対応状況を集計を表 7 に示す．HMAC については強度の高い SHA384, SHA512 の対応が 2017 年以前発売の機種では少ないとわかる．Android OS バージョン別の集計を表 8 に示す．Android OS 9 系ではすべてのアルゴリズムで対応率は高い．それ以前の Android OS 7 系・8 系では，HMAC のうち SHA256 を除くハッシュアルゴリズムで対応率が低かった．

4.2　Keystore API の利用状況

Android アプリを静的解析して得られた，表 5 に示した Keystore API の利用状況の結果を表 9 に示す．`main` および `other` は，Keystore API の利用を判定する文字列パターン (表 5 における $(p1 \mid p2)$) がアプリのメイン機能側で発見されたか否かを示している．`main` ∪ `other` は，ア

表 5　Keystore API 利用の文字列パターン
Table 5　String patterns of Keystore API.

	パターン	説明
$p1$	KeyGenerator.getInstance	共通鍵系 (AES,HMAC) の鍵生成時に呼び出し
$p2$	KeyPairGenerator.getInstance	公開鍵系 (RSA,ECDSA) の鍵生成時に呼び出し

表 6　暗号アルゴリズム別の対応機種数 (N=71)
Table 6　Number of adopted devices by crypto algorithms (N=71).

		対応	非対応
RSA		71(100%)	0(0%)
ECDSA		70(98.6%)	1(1.4%)
AES		70((98.6%)	1(1.4%)
HMAC	SHA1	65(91.5%)	6(8.5%)
	SHA224	60(84.5%)	11(15.5%)
	SHA256	70(98.6%)	1(1.4%)
	SHA384	60(84.5%)	11(15.5%)
	SHA512	60(84.5%)	11(15.5%)

プリのソースコードにおいて少なくとも 1 つ以上発見されたということを示している．

結果から，アプリに明示的に Keystore が実装されているものは，659 アプリ（3.7%）であり，アプリでの利用状況はそれほど多くない結果である．また，13,659（76.9%）のアプリが Keystore API を利用していることがわかるが，`main` と分類されたアプリのメイン機能側での Keystore API 利用よりも，`other` と分類されたサードパーティライブラリ側での利用が多い．

我々は，`other` と分類されたサードパーティライブラリ等の利用状況を追加調査した．図 3 に，`other` と分類された Keystore API 利用のパッケージ構造を，アプリでの利用数が多いもの上位 20 を示す．最もアプリでの利用が多い Keystore API が利用されているライブラリは Firebase[12] の 9246 アプリであった．その他，アプリでの利用が多いものは Google Play Services[13] のライブラリ群である．

4.3　デバイス側の対応傾向

今回の調査で対象とした国内で流通するスマートデバイスはいずれも RSA 暗号のハードウェアベース暗号鍵管理に対応していた．したがって，すべての機種でハードウェアベース暗号鍵管理機能は利用可能だと言える．一方で，楕円曲線暗号や AES，HMAC に関しては非対応の機種が存在した．特に HMAC に関してはばらつきが多い状況にあるものの，HMAC で用いるハッシュアルゴリズムについて，HMAC-SHA256 は 1 機種を除きすべて対応していた．HMAC-SHA256 は電子政府推奨暗号リスト [14] に含まれており，現時点で十分な強度を確保したアルゴリズムに対応する機種が多いと言える．RSA 以外のアルゴリズムに関しては，高い割合で利用可能と言えるもののすべての機種で利用可能であると期待できないと言える．

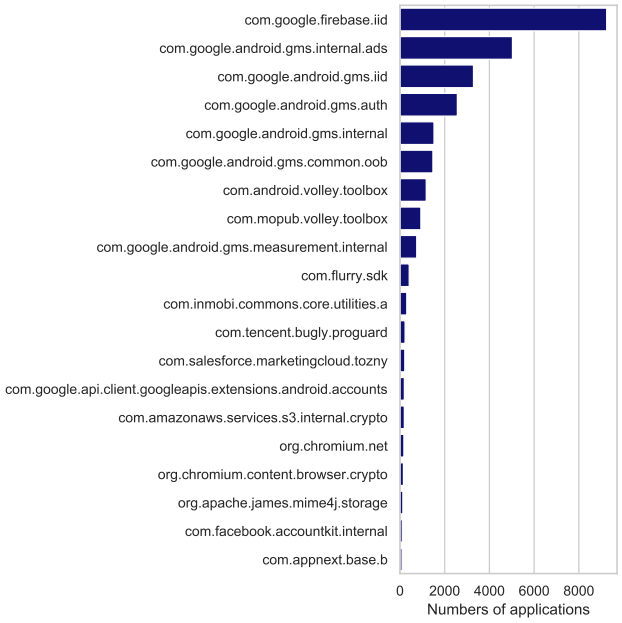

図 3　ライブラリを利用してるアプリ数
Fig. 3　Numbers of apps using the libraries.

以上を踏まえると，日本国内で流通するスマートデバイスにおいてはハードウェアベース暗号鍵管理の対応率は高いものの，利用可能な暗号アルゴリズムに制約がある状況であると言える．

4.4　アプリ側の利用傾向

4.2 節の結果から Keystore の利用率は高いが，Keystore API を利用するライブラリを多くのアプリが導入していることに起因することによるものと考えられる．アプリ開発支援サービスである Firebase や Google Play Services は多くのアプリに利用されており，これらライブラリによって生成された認証用暗号鍵などは，デバイス側が対応していればハードウェアベースで安全に管理される．しかし，メイン機能側における Keystore API の利用率は低く，アプリ開発者が明示的に暗号鍵を利用してアプリを開発している状況とは言い難い．

4.5　Android プラットフォームにおける鍵管理の状況

Android プラットフォーム上で，暗号鍵管理を担う Keystore は多くのデバイスにおいて，ハードウェアベースの強固な保護が可能となっている．一方で，暗号アルゴリズムによって対応状況にばらつきが見られ，特定のアルゴリ

表 7　デバイス発表時期別の対応機種数
Table 7　Number of adopted devices by released year.

		2015	2016	2017	2018	2019
RSA		1(100%)	3(100%)	18(100%)	30(100%)	18(100%)
ECDSA		1(100%)	3(100%)	17(94.4%)	30(100%)	18(100%)
AES		1(100%)	3(100%)	17(94.4%)	30(100%)	18(100%)
	SHA1	1(100%)	3(100%)	13(18.2%)	30(100%)	18(100%)
	SHA224	1(100%)	1(33.3%)	10(55.6%)	30(100%)	18(100%)
HMAC	SHA256	1(100%)	3(100%)	17(94.4%)	30(100%)	18(100%)
	SHA384	1(100%)	1(33.3%)	10(55.6%)	30(100%)	18(100%)
	SHA512	1(100%)	1(33.3%)	10(55.6%)	30(100%)	18(100%)

表 8　OS バージョン別の対応機種数
Table 8　Number of adopted devices by OS versions.

		7 系	8 系	9 系
RSA		3(100%)	15(100%)	53(100%)
ECDSA		3(100%)	14(93.3%)	53(100%)
AES		3(100%)	14(93.3%)	53(100%)
	SHA1	2(66.7%)	14(93.3%)	49(92.5%)
	SHA224	2(66.7%)	10(66.7%)	48(90.6%)
HMAC	SHA256	3(100%)	14(93.3%)	53(100%)
	SHA384	2(66.7%)	10(66.7%)	48(90.6%)
	SHA512	2(66.7%)	10(66.7%)	48(90.6%)

表 9　API 利用状況調査の結果 (N=17,757)
Table 9　Results of API usages (N=17,757).

`main` ∪ `other`	`main`	`other`	`main` ∩ `other`
13,659	659	13,565	565
(76.9%)	(3.7%)	(76.4%)	(3.2%)

ズムにおいてはハードウェアによる保護が受けられない状況が存在すると言える．また，アプリ側では Keystore を用いた暗号鍵管理がライブラリを中心に広く実装されており，特にアプリ開発者からは暗黙的な形でデバイス上で Keystore が利用されている状況が伺える．

一部のアルゴリズムにおいてハードウェアの保護が受けられないことから，特定の条件下ではあるものの，開発者が認識していないクレデンシャル管理の侵害リスクが存在していると考えられる．また，能動的に選択したい開発者にとってもいくつかの制約が存在すると考えられる．

4.6　本調査の制限事項

今回の調査結果に基づき，Android OS を搭載したスマートデバイス一般のハードウェアベース暗号鍵管理の対応率を判断する場合，今回の調査対象としたデバイス集団と実際に普及しているデバイス集団の差異から対応率に差異が存在すると考えられる．調査対象としたデバイス集団には，各デバイスメーカの市場シェアや個別のデバイスの販売台数の割合などを加味していないためである．実際のアプリ開発において，今回調査したハードウェアベース暗号鍵管理機能が求められる場合は，事前にアプリの動作対象とするデバイス群を明確にし，実装状況の確認が求められると考えられる．

アプリ側の利用状況では，Keystore API の呼び出し箇所を基に利用形態を調査している．そのため，実装されているものの実際の稼働時に利用されていない場合や実際のアプリ上のクレデンシャル管理との連携状況は加味していない．また，今回の調査では個々のデバイスの実装方式やアプリ側で利用しているアルゴリズムは調査対象としていない．

5.　調査結果に対する考察

5.1　暗号アルゴリズム選択の制約

今回の調査では ECDSA, AES ならびに HMAC においてスマートデバイスの対応状況にばらつきが確認された．これらアルゴリズムを用いてクレデンシャル管理を実装しようとするサードパーティアプリ開発者が当該アルゴリズムを選択できない状況が生じていると言える．Android OS の Keystore は，今回調査した暗号アルゴリズムを仕様上サポートしており，アプリはいずれの暗号アルゴリズムも採用し，実装することができる．一方，個別のデバイスがアプリが実装した暗号アルゴリズムのハードウェアベースでの暗号鍵管理に対応していない場合，当該アルゴリズムの処理はソフトウェアベースで実行される．そのため，アプリ開発者がアプリが実行されるすべてのデバイスにおいてハードウェアベースの暗号鍵管理を必要とする場合，ECDSA や AES, HMAC は採用できない．特に他サービスやデバイスとの連携や設計上の制約において，RSA が採用できない場合，アプリ開発者はスマートデバイス上での安全な鍵管理を断念するか対応策の検討が求められる．

また，各デバイスでの対応状況が事前に確認できない点が，アプリ開発者にとって負担になると言える．今回判別に利用した Keystore API は，生成された暗号鍵の管理状況を確認するメソッドである．したがってスマートデバイス上で実際に暗号鍵を生成するまではアプリ開発者からどのように当該デバイスで暗号鍵が管理されるかを判断することができない．Google 社は同社のアプリケーション配

信プラットフォーム上で，アプリ配信対象のデバイスのスペックの確認やスペックに基づく配信制限機能を備えている．しかし，事前確認できるスペックや制限機能には，今回調査した暗号鍵管理機能の実装状況は含まれていない [15]．そのため，スマートデバイス向けアプリの設計時点においては，選択可能な暗号アルゴリズムに大きな制約がかかってしまう点が課題であると言える．

5.2 鍵管理機能対応率向上の長期化

スマートデバイスのライフサイクルは，スマートデバイスそのものの性能向上とともに長期化する傾向にある．これに伴い，ハードウェアベース暗号鍵管理に対応しないデバイスが長期に渡り市場に存在し続ける可能性が高い．暗号アルゴリズムの選択に対する制約も長期化すると考えられる．

スマートデバイスの多くは，ソフトウェアやファームウェア更新機能を備えている．しかし，アップデートの提供はデバイスの発売後数年で打ち切られることが多い．スマートデバイスのハードウェア側の鍵管理機能の対応が可能であっても，デバイスメーカによりソフトウェア側の対応がなされないと非対応の状況は改善されない．Google 社が調査した Android OS のバージョン別利用シェア [16] では，今回の調査対象としなかった Android 6.0 以前の端末が 42.1%を占めている．Android OS 6.0 以前はハードウェアベース暗号鍵管理の実装はデバイスメーカーの任意対応とされているため，一定数のシェアを持つ販売済み端末が非対応のままユーザに利用され続ける可能性がある．

5.3 暗号アルゴリズム移行の困難性

暗号アルゴリズムは，計算機性能の向上や新たな攻撃法の発見などにより危殆化のリスクが存在する．このようなリスクに対し，より強度の高い暗号アルゴリズムへの移行が行われることで対応している．暗号アルゴリズムの移行には，現在のアプリやデバイスで保護しているクレデンシャルの移行などを伴うため，移行にかかる時間は長期化する傾向にある．そのため，危殆化リスクが生ずると想定される時期から遡り，計画的に移行を実施することが求められる．

今回調査した鍵管理機能は 5.1 節で述べたように実際に鍵生成を実施しない限り，アプリ開発者やエンドユーザからはデバイスが備える鍵管理機能が把握できない．また，アプリ側の利用状況からそもそも暗号を利用している認識がない開発者も存在すると考えられ，アルゴリズム移行の啓発やライブラリの対応などのプロセスも含めた場合，移行プロセスの長期化につながることが想定される．特にアプリ開発者が移行に先立ち，利用状況やリスクの把握へのコストが増大すると考えられる．

5.4 研究倫理について

本稿では 3 章で述べた目的を達成するため，市販されるデバイス実機に対する調査ならびにアプリの静的解析を行った．研究倫理への対応として，コンピュータセキュリティシンポジウム（CSS）2019 の「サイバーセキュリティ研究における倫理的配慮のためのチェックリスト [17]」を利用した．本稿の各調査における細かな倫理的配慮事項は，下記の通りである．

5.4.1 デバイス調査に関する倫理的配慮

今回の調査で対象とした市販デバイスに関して，調査対象群の網羅性を示す観点からデバイスメーカの具体名を表 3 にて示している．個別のデバイスの調査結果（対応状況）に関しては，特に非対応機種への不利益を最小化する観点から本稿での記載は行っていない．また，結果の記載では個別の機種の判別に至らぬよう機種数のみを示す形式を採用した．

5.4.2 アプリ調査に関する倫理的配慮

アプリ調査では，文献 [11] と同様の倫理的配慮を行った．

6. 関連研究

スマートデバイスでのクレデンシャル保護や暗号利用においては，デバイス側を対象としたものとして，Cooijmans ら [18] は，Android プラットフォームで利用可能な暗号鍵管理の手法について分析している．7 機種の対応状況を調査しているが，調査対象アルゴリズムは RSA のみとなっている．Android OS 8.0 で実装された Keystore Key Attestation 機能では，デバイスに搭載された暗号鍵を用いて外部から Keystore の構成を検証することができる [7]．この機能に対応したスマートデバイスは，検証用の暗号鍵や証明書を格納した状態で出荷される．Daniel は，Attestation 機能を利用したデバイス検証アプリ [19] を開発しており，各デバイスの検証用証明書を開発したアプリを通じて収集し，公開している [20]．検証用暗号鍵はハードウェア内に格納されているため，検証用証明書が収集されたデバイスはハードウェアベース鍵管理に対応したデバイスとみなすことができ，52 機種の証明書が収集されている．一方で，Attestation 機能の対応状況から個別のデバイスのアルゴリズム単位での対応状況を確認することはできない．

アプリの実装を対象とした調査には，Egele らの調査 [21] や Gao らの調査 [22] がある．これらは Android OS 向けに配布されているアプリのうち，暗号 API を用いるアプリに含まれる誤用とされる実装を調査している．調査対象の暗号 API は Java ベースの暗号ライブラリである Bouncy Castle[23] や JCA（Java Cryptography Architecture）[24] であり，本稿で調査した Keystore とは対象が異なっている．

7. まとめ

本稿では，スマートデバイスがクレデンシャルの保存先

となっている状況を踏まえ，クレデンシャルを強固に保護する手段であるハードウェアベース暗号鍵管理機能の利用状況を調査した．調査対象としてデバイス側とアプリ側の双方を対象とし，前者は日本国内で流通する Android OS を搭載したスマートデバイス 71 機種とし，複数の暗号アルゴリズムの対応状況を確認した．その結果，調査対象のすべての機種で RSA 暗号の鍵管理に対応している．一方，楕円曲線暗号や対称鍵暗号系では対応状況にばらつきが確認された．後者では，日本向けに配信されるアプリ 17,757 本を対象に，暗号鍵管理 API の呼び出しの有無を調査し，76.9%のアプリで実装されていることを確認した．今後の課題として，個々のデバイスの実装方式調査や鍵生成前に対応状況を確認可能な方式の提案が挙げられる．

参考文献

[1] Google LLC: Android Keystore system, Google LLC (online), available from ⟨https://developer.android.com/training/articles/keystore⟩ (accessed 2019-08-11).

[2] Apple Inc.: Keychain Services, Apple Inc. (online), available from ⟨https://developer.apple.com/documentation/security/keychain_services⟩ (accessed 2019-08-11).

[3] Google LLC: Jelly Bean — Android Developers, Google LLC (online), available from ⟨https://developer.android.com/about/versions/jelly-bean⟩ (accessed 2019-08-11).

[4] Cooijmans, T., de Ruiter, J. and Poll, E.: Analysis of secure key storage solutions on android, *Proceedings of the 4th ACM Workshop on Security and Privacy in Smartphones & Mobile Devices*, ACM, pp. 11–20 (2014).

[5] ARM Limited.: ARM Security Technology Building a Secure System using TrustZone Technology (2009).

[6] Android Open Source Project: Hardware-backed Keystore, Android Open Source Project (online), available from ⟨https://source.android.com/security/keystore⟩ (accessed 2019-08-11).

[7] Willden, S.: Android Developers Blog: Keystore Key Attestation, https://android-developers.googleblog.com/2017/09/keystore-key-attestation.html (2017).

[8] Xin, X.: Titan M makes Pixel 3 our most secure phone yet, Google LLC (online), available from ⟨https://www.blog.google/products/pixel/titan-m-makes-pixel-3-our-most-secure-phone-yet/⟩ (accessed 2019-08-11).

[9] Cohen, E.: How can Snapdragon 845 guard your smartphone data like a vault?, Qualcomm Technologies, Inc. (online), available from ⟨https://www.qualcomm.com/news/onq/2018/03/08/how-can-snapdragon-845-guard-your-smartphone-data-vault⟩ (accessed 2019-08-13).

[10] Google LLC: KeyInfo, Google LLC (online), available from ⟨https://developer.android.com/reference/android/security/keystore/KeyInfo.html#isInsideSecureHardware⟩ (accessed 2019-08-11).

[11] 坂本一仁，藤本裕之，松永昌浩：WebView 搭載 Android アプリケーションにおけるユーザへの Cookie コントロール機能提供状況の調査，コンピュータセキュリティシンポジウム 2019 論文集 (2019).

[12] Google LLC: Firebase, Google LLC (online), available from ⟨https://firebase.google.com/⟩ (accessed 2019-08-21).

[13] Google LLC: Overview of Google Play Services, Google LLC (online), available from ⟨https://developers.google.com/android/guides/overview?hl=ja⟩ (accessed 2019-08-21).

[14] 経済産業省総務省：電子政府における調達のために参照すべき暗号のリスト (2016).

[15] Google LLC: Supported devices - Play Console Help, Google LLC (online), available from ⟨https://support.google.com/googleplay/android-developer/answer/6154891?hl=en&ref_topic=7071529⟩ (accessed 2019-08-11).

[16] Google LLC: Android Developers Dashboard, Google LLC (online), available from ⟨https://developer.android.com/about/dashboards/⟩ (accessed 2019-08-11).

[17] コンピュータセキュリティシンポジウム（CSS）2019 研究倫理委員会：サイバーセキュリティ研究における倫理的配慮のためのチェックリスト (2019).

[18] Cooijmans, T., de Ruiter, J. and Poll, E.: Analysis of Secure Key Storage Solutions on Android, *Proceedings of the 4th ACM Workshop on Security and Privacy in Smartphones & Mobile Devices*, SPSM '14, New York, NY, USA, ACM, pp. 11–20 (online), DOI: 10.1145/2666620.2666627 (2014).

[19] Micay, D.: Device Integrity Monitoring, (online), available from ⟨https://attestation.app/about⟩ (accessed 2019-08-21).

[20] GrapheneOS: GrapheneOS/Attestation Samples, GrapheneOS (online), available from ⟨https://github.com/GrapheneOS/AttestationSamples⟩ (accessed 2019-08-20).

[21] Egele, M., Brumley, D., Fratantonio, Y. and Kruegel, C.: An Empirical Study of Cryptographic Misuse in Android Applications, *Proceedings of the 2013 ACM SIGSAC Conference on Computer & Communications Security*, CCS '13, New York, NY, USA, ACM, pp. 73–84 (online), DOI: 10.1145/2508859.2516693 (2013).

[22] Gao, J., Kong, P., Li, L., Bissyandé, T. F. and Klein, J.: Negative Results on Mining crypto-API Usage Rules in Android Apps, *Proceedings of the 16th International Conference on Mining Software Repositories*, MSR '19, Piscataway, NJ, USA, IEEE Press, pp. 388–398 (online), DOI: 10.1109/MSR.2019.00065 (2019).

[23] Legion of the Bouncy Castle Inc: bouncycastle.org, Legion of the Bouncy Castle Inc. (online), available from ⟨http://bouncycastle.org/⟩ (accessed 2019-08-21).

[24] Oracle: Java cryptography architecture, Oracle (online), available from ⟨https://docs.oracle.com/javase/7/docs/technotes/guides/security/crypto/CryptoSpec.html⟩ (accessed 2019-08-21).