Androidアプリケーションにおける電子署名の大規模調査

吉田 奏絵[1]　今井 宏謙[1]　芹沢 奈々[2]　森 達哉[2]　金岡 晃[1]

概要：Android アプリケーションのアプリケーションパッケージ（APK）には電子署名が付与されている。本稿では APK に付与された電子署名について、117 万のアプリケーションを対象に電子署名の状況を調査した。調査の結果、鍵長の短い RSA 鍵の利用や、脆弱なハッシュ関数 MD5 の利用など、多くのアプリに問題があることがわかった。また Android 特有の電子署名の扱い方により、脆弱なハッシュ関数等によりセキュリティとプライバシの両面で脅威が存在することがわかった。調査結果で判明した事実に加え、脅威の現実性についても検討を行う。

キーワード：システム, Android, 電子署名

Abstract: In this paper, we perform large-scale survey for digital signatures to Android's Application Package (APK) with about 1.17 million applications. We found that there are short RSA keys and weak hash algorithms like MD5. Thus, we discuss about realistic threat on android applications by its specific digital signature usage.

Keywords: System, Android, Digital Signature

1. はじめに

Android OS やその上で動くアプリケーション（以後、Android アプリ）群のセキュリティの担保とプライバシの保護が様々な角度から議論がされている。Android OS が搭載される端末は、スマートフォンに代表されるように利用者に携帯される特性を持つことが多く、利用者個人の情報が集約されることに加え、マイクやカメラ、GPS 等の様々なセンサの搭載により利用者周辺のコンテキスト取得が容易になり、利用者個人のプライバシに関連する情報も多く取得されるなど、従来の PC 等の端末以上のセキュリティとプライバシの議論が求められる。

セキュリティの担保の 1 つの方策として、Android アプリでは電子署名を利用してアプリケーションパッケージ（APK）が持つファイル群の完全性を保証している。電子署名等の暗号技術は、Android のみならず、Web サーバでの利用や各種機器での利用など広範な利用が進んでいが、デフォルトの設定のまま使うことや特定の環境下で脆弱な鍵が生成されるなど、実装や運用・管理面で問題となるケースがある。Android アプリの APK に対する電子署名についても、実装や運用・管理面において問題が生じる可能性があるが、これまでそういった調査を大規模かつ網羅的に行ってきた研究は著者らの知る限り存在していなかった。

そこで本稿では、APK に付与された電子署名について、Google 社の Google Play マーケット上の 117 万を超えるアプリ群を対象に、電子署名の状況を調査した。APK は PKCS#7 方式で電子署名が付与されており、PKCS#7 のデータを抽出することで、署名に利用された暗号アルゴリズムと証明書の内容、そして APK の署名に利用されたハッシュ関数の情報を得て、さまざまな分析を行う。

分析の結果、Google Play の制約や Google 社の推奨により 99%以上の証明書が有効期限が 25 年以上と長期間のものにわたることがわかった。そして長期間有効な証明書を利用している一方で、利用されている暗号アルゴリズムと鍵長が RSA512 ビットや RSA1024 ビットなど長期間の安全性が期待できないものが使われていることも判明した。RSA の 1024 ビット鍵は 48.13%のアプリで利用されているなど多くのアプリが長期間の安全性に耐えなない暗号を利用している。さらに、衝突攻撃が現実的な脅威となっている MD5 の利用、また移行が勧められている SHA-1 が広範囲に利用されていることなど、暗号の運用面では課題を抱えていることが明らかになった。MD5 は全体の 4.49%、SHA-1 は 95.45%と、ほぼすべてのアプリが MD5 や SHA-1

[1] 東邦大学
Toho University
[2] 早稲田大学
Waseda University

など現在での利用、とくに長期間のデータ保証には適さないアルゴリズムが使われていた。

加えて、素因数分解が成功することによる秘密鍵の取得でどのような脅威が存在するか、またハッシュ関数の衝突を起こすことでのような脅威が起こるかを考察した。特にハッシュ関数の衝突による署名の偽造については、Android アプリ特有の脅威が存在することを示した。そしてそれぞれの脅威の現実性についても議論を行った。

2. 関連研究

Android アプリやそのマーケットの調査については、大規模な調査研究が多く行われている。この 2 年間だけに限っても、渡邉らによる研究 [1] や、Ren らによる研究 [2] が挙げられる。また、マーケットの解析に特化した国際会議 International Workshop on App Market Analytics が 2016 年 11 月に開催されるなど、研究が盛んにされている。

電子署名や暗号実装に関する調査研究も多くがされている。代表的なのが Heninger らによる調査であろう [3]。Heninger らの研究では、ネットワーク機器が生成する鍵に情報量（エントロピー）が不十分に生成される特徴があることや、RSA の公開鍵などで素因数を使いまわすものがあるなど、実装と運用において暗号利用に大きな問題点があることをしめしていた。最近でも、Svenda らが RSA の公開鍵を分析することで生成したツールのグループを推測可能であることを示すなどの研究が行われている [4]。

Android に関連するものとしては、複数の Android アプリで SSL/TLS 証明書の検証に不備があることが 2014 年に CERT/CC により報告されるなど、適切な利用がされていない状況が指摘されている [5]。

こういった研究は、リスクに対する対策の研究と同時に、脅威の正確な把握として非常に重要であり、こういった調査結果をもとにさらなる効果的な対策や評価につながっている。少しテーマが離れるが、パスワード研究において Weir らが分析した結果をもとにパスワード推測攻撃が飛躍的な向上をした一方で [6]、そういった推測攻撃への耐性を指標として対策を検討し評価する研究がその後多くされていることは代表的な例と言えよう [7], [8], [9], [10]。

3. Android APK の電子署名

3.1 署名の仕組み

Android アプリではインストールするすべてのアプリケーションに対して電子署名が付与される。また署名に利用された証明書を必要とする。この証明書の秘密鍵は、APK の開発者が所持する。

アプリの改ざん防止・検知のために、AndroidManifext.xml を除くリソースファイル等のファイルそれぞれのハッシュ値を計算し、ファイルのパスとハッシュ値が MANIFEST.MF に記載される。さらに、MANIFEST.MF のハッシュ値を計算し、MANIFEST.MF に記載されている各ファイルのハッシュ値を計算し、ファイルのパスとハッシュ値が SF ファイルに記載される。SF ファイルに記載されるハッシュ値計算は、各ファイルのハッシュ値を計算するにあたってハッシュ関数の内部状態を初期化せず、その前のファイルのハッシュ値を求めた際の内部状態を保持したまま次のファイルのハッシュ値を計算する。SF ファイルは拡張子が.SF となったファイルであり、CERT.SF という名称が一般的に使用されている。SF ファイルに RSA 署名鍵（または DSA 署名鍵）で署名し、PKCS#7 形式で保存したものが拡張子.RSA のファイル（または.DSA の拡張子）となる。また、.RSA、.DSA ファイルには署名者の公開鍵の情報を含んでいる。

これらの証明書情報は、APK の内部フォルダである「META-INF」に格納される。

3.2 信頼の仕組み

APK ファイルへの署名に用いられている証明書は、広く信頼された認証局（CA）から発行をされていなくてよく、開発者あるいはその組織が発行した自己署名証明書を利用しても問題がない。よって、APK の利用者である Android 端末利用者は電子署名単体としてはその発行元の信頼確認をできない。開発者あるいはその組織自体の信頼情報を利用者は持っていないためである。

Google 社の Play マーケットでは、その点を Play マーケットへの登録作業をすることで担保をしていると考えられる。Play マーケットでアプリを公開するにあたり、開発者情報を登録しなければならず、Google 社の審査によりその公開の有無が判断をされる。Play マーケット上にあるアプリは Google 社の審査を経たものであることを利用者は認識し、それをダウンロードし利用する。

つまり APK への署名は、署名がされている事実が重要であり、だれが署名したかは Play ストアさらには Android アプリとしては重要ではなく、データの完全性保証が主の用途であるといえる。そういった点は、APK への署名に用いる証明書の発行についての文書でも見ることができる [11]。そこでは証明書の有効期限を 25 年以上にすることが指定されている。利用される暗号のアルゴリズムは鍵のサイズにかかわらず有効期限が 25 年以上が指定されていることはそれを裏付けていると言えよう。3.4.2 節でも述べるが、そのほかにも 10000 日以上への推奨や、2033 年 10 月 22 日までの有効期限を必須とする指定がある。

3.3 署名を利用したアプリ間のアクセス制御

Android アプリはカメラや連絡帳などの端末の機能やデータの利用が制限されており、これらへのアクセスを許可するために AndroidManifest.xml ファイルにパーミッションを宣言する。パーミッションには 4 つの保護レベ

ル（normal/dangerous/signature/signatureOrSystem）が定められており、そのパーミッションの利用範囲を制限している。このうち、signature と signatureOrSystem は、パーミッションを定義したアプリと同じ署名を持つアプリに、そのパーミッションの使用を許可している。

パーミッションは製作者が独自に作ることもでき、その時に製作者が保護レベルを指定することが可能である。

3.4 その他

3.4.1 暗号アルゴリズムの選択

Android アプリでは、署名に用いる公開鍵アルゴリズムは RSA と DSA の 2 つのみサポートされている。

3.4.2 有効期間

Google 社は公開鍵の有効期限を 10000 日以上（27.4 年以上）を推奨している。また、公開鍵は 2033 年 10 月 22 日までの有効期間が必要で、有効期間がこの日付以前に期限切れになるキーで署名されたアプリケーションは、アップロードできないとされている [12]。

4. Android アプリケーションの電子署名状況調査

4.1 調査内容

本研究では、Android アプリケーションに付与された電子署名の現状を調査するために、PlayDrone で集められた APK ファイルのデータセットについて分析を行った。

PlayDrone は 2014 年に Viennot らにより開発されたクローラである [13]。クローラだけでなく、Viennot らは PlayDrone で収集されたデータセットを Archive.org にて公開しており、今回の分析では公開されているデータセットに含まれていた 1,177,599 個の APK を利用した。これらの APK は Google Play マーケットを対象に収集されたもので、収集の時期詳細は不明だが、2014 年 8 月にアップロードがされている。

解析においては、各 APK ファイルより META-INF フォルダ内の.RSA あるいは.DSA を取得し、証明書情報を取得と、APK への署名に利用したハッシュ関数情報を取得する。取得した証明書情報等から各情報をさらに抜き出し、分析に利用した。証明書から抽出した情報は以下の通りである。

- 公開鍵が RSA の場合
  - 署名者情報
  - 被署名者情報
  - 署名アルゴリズム
  - 鍵長
  - RSA の modulus 値 (n)
  - RSA の exponent 値 (e)
  - 有効期限（開始）
  - 有効期限（終了）
  - APK のパス名（アプリ名の取得）

- 公開鍵が DSA の場合
  - 署名者情報
  - 被署名者情報
  - 署名アルゴリズム
  - 鍵長
  - DSA の pub 値
  - DSA の P 値
  - DSA の Q 値
  - DSA の G 値
  - 有効期限（開始）
  - 有効期限（終了）
  - APK のパス名（アプリ名の取得）

また PKCS#7 データからは、署名時に利用されたハッシュ関数の OID を抽出した。

4.2 調査結果

4.2.1 署名に用いられた公開鍵暗号

APK ファイルの電子署名に用いられた公開鍵暗号のアルゴリズムとその鍵長についての調査結果を表 1 に示す。RSA1024 は公開鍵暗号アルゴリズムとして RSA が使われ鍵長が 1024 ビットであることを示している。RSA の利用は 1,151,312 個であり、DSA の利用は 26,803 個であった。なお、いくつかの APK については複数の電子署名が付与されているため、鍵の総数は APK 総数を超えている。

表 1 署名に用いられた鍵アルゴリズムと鍵長

鍵長と種類	個数	割合
RSA1024	567,055	48.13%
RSA2048	580,134	49.24%
RSA4096	3,077	0.26%
DSA1024	26,697	2.27%
その他	1,152	0.10%

RSA では 2048 ビットの鍵が最も多く使われており、全体の 49.24%を占めている。次いで RSA の 1024 ビット鍵が全体の 48.13%を占めている。RSA では 86 種類の鍵長が使われており、最も短いもので 512 ビット、最も長いもので 16384 ビットであった。512 ビットの RSA 鍵を用いた APK は 223 個あり、16384 ビットの RSA 鍵を用いた APK は 6 個であった。

DSA では 1024 ビットの鍵が最も多く使われており、全体の 2.27%を占めている。なお DSA だけで見ると、その 99.6%が 1024 ビットの鍵であった。DSA では 4 種類の鍵長が使われており、最も短いもので 512 ビット、最も長いもので 2048 ビットであった。512 ビットの DSA 鍵を用いた APK は 6 個あり、2048 ビットの DSA 鍵を用いた APK は 2 個であった。

表 2　証明書の署名に使用されていた署名アルゴリズム

署名アルゴリズム	個数	割合
md5WithRSAEncryption	18,380	1.60%
sha1WithRSAEncryption	717,503	60.90%
sha256WithRSAEncryption	415,238	35.25%
dsaWithRSA	26,803	2.28%
その他	193	0.02%

表 3　自己署名証明書の割合

自己署名かどうか	個数	割合
自己署名	1,177,185	99.92%
自己署名でない	931	0.08%

4.2.2　証明書の署名に用いられた署名アルゴリズム

各 APK の証明書に利用されていた署名アルゴリズムの調査結果を表 2 に示す。md5WithRSAEncryption は署名に RSA が用いられ、証明書内容のハッシュ値取得にハッシュ関数として MD5 が用いられていることを示している。調査結果から、sha1WithRSAEncryption が 60.90%と最も多く利用されていることが分かった。次いで 35.25%で sha256WithRSAEncryption が利用されていた。特徴的なのが、1.60%の割合で md5WithRSAEncryption が利用されていたことであろう。個数としては 18,380 の APK がハッシュ関数として MD5 を用いた署名がされていた。MD5 の利用については、衝突攻撃が現実的な時間で実現可能かとこから電子署名の利用としては望ましくなく、例えば CRYPTREC では政府推奨暗号リストから外れている。SHA-1 については衝突攻撃等の解析が現実的な時間で実現できるかについてはまだ明らかになっていないが、強度については低下が指摘されているため SHA-256 等への移行が進められている。

4.2.3　自己署名証明書の利用

APK ファイルへの署名に用いられる証明書は、自己署名証明書であることは問題がない。それらの実態について調査した結果を表 3 に示す。99.92%とほぼすべての APK が自己署名証明書を用いて署名されていることがわかった。3.2 章において、APK への署名は署名の事実が重要であり誰が署名したかは重要ではないと指摘したが、それを支持する内容と言えよう。

4.2.4　証明書の有効期間

APK の署名に使われる証明書の有効期間については、Google 社のドキュメントにおいて 10000 日以上を推奨し、有効期限が 2033 年 10 月 22 日以降までに達していない証明書の鍵で署名されたアプリは Google Play マーケットにアップロードできないとなっている。

証明書の有効期間について調査した結果を表 4 に示す。有効期間の分類は Google 社の推奨する値によって行った。Google 社は 10000 日以上と 25 年（9125 日）以上と 2 つの推奨を別のドキュメントで行っている。

表 4　証明書の有効期間

有効期間（日数）	個数	割合
9125 日未満	5,365	0.46%
9125 日以上 10000 日未満	276,011	23.43%
10000 日以上	896,742	76.12%

表 5　MD5 利用証明書の有効期間

有効期間（日数）	個数	割合
9125 日未満	46	0.25%
9125 日以上 10000 日未満	2,370	12.89%
10000 日以上	15,964	86.86%

表 6　RSA の指数 e の分布

e の値	個数	割合
17 (0x11)	309	0.03%
3 (0x3)	1,187	0.10%
35 (0x23)	1	0.00%
65537 (0x10001)	1,149,812	99.87%

全体の 0.46%、5,365 個の APK が 9125 日未満の有効期間となっていた。最短の有効期間は 90 日であり、該当する APK は 5 個が存在した。しかし、この証明書は単体では用いられておらず、有効期間が 10000 日以上の証明書と合わせて署名がされていた。また他の 9125 日未満の証明書のものも、多くは有効期限が 2033 年 10 月 22 日以降になるように設定されたものであった。

もっとも多かったのは 10000 日以上の有効期間を持つ証明書であり、全体の 76.12%を占めた。またその中で最長の有効期間は 2,932,896 日（8035.3 年）であった。

また、MD5 を利用している APK 群に限った有効期間の分布を表 5 に示す。そのほとんどが 25 年以上の有効期間を持つことがわかる。さらにその中で有効期間が 36500 日（100 年）以上のものが 3,958 個あった。

4.2.5　RSA 公開鍵の e の分布

RSA 暗号では、公開鍵として e, n の 2 つのパラメータが公開される。e で利用されている値の分布を表 6 に示す。65537 が最も多く利用されており 99.87%を占める。一方で、1187 個の APK で $e = 3$ が用いられていることがわかった。

4.2.6　RSA 公開鍵の n の素因数分解

Heninger らの研究 [3] では、ネットワーク機器に利用されている鍵の脆弱性についてを最大公約数（GCD）を求めることで高速に割り出していた。Heninger らはそれらの実験に用いたプログラムを公開しており、本研究ではそのプログラム（以降、fastgcd と呼ぶ）を用いた分析も行った。

分析は 2 つのタイプで行った。まず収集した APK 群の公開鍵パラメータの n を用いて GCD の計算を行った。また、収集した APK 群だけでなく、Rapid7 社が実施している Project Sonar で収集した SSL/TLS 証明書の公開鍵を用いて GCD 計算を行った。Project Sonar のデータ

表 7　APK の署名に使用されていた署名アルゴリズム

署名アルゴリズム	個数	割合
SHA-1	1,123,779	95.45%
SHA-256	680	0.06%
MD5	52,866	4.49%

表 8　証明書で利用されている暗号アルゴリズムと APK 署名に利用されているハッシュ関数

	MD5	SHA-1	SHA-256	不明
MD2	0	74	0	0
MD5	4450	13798	18	114
SHA-1	30631	690107	149	4032
SHA-256	17532	394416	513	2444
SHA-384	0	25	0	0
SHA-512	0	86	0	0

では、2016 年 7 月 25 日と 2016 年 8 月 1 日に収集された 1,389,639 個の証明書データを用いた。

fastgcd 実施の結果、最大公約数の発見はなかった。このことから Heninger らの研究であったような生成プロセスにおける脆弱な鍵発行は、Android アプリの開発環境においては見つからなかったといえる。

4.2.7　APK の署名に用いられたハッシュ関数

各 APK の署名に利用されていた署名アルゴリズムの調査結果を表 7 に示す。SHA-1 の利用が 95.45%と最も多い。次いで MD5 が 4.49%と続く。MD5 の利用率は証明書での利用率（1.60%）よりも多くなっていることからもわかるように、証明書で利用されているハッシュ関数と APK に対して利用しているハッシュ関数はか必ずしも一致していない。表 8 は、RSA 暗号に関して証明書で利用されているハッシュ関数と APK 署名に用いられているハッシュ関数のクロス集計表である。APK への署名に MD5 を用いているうち、証明書で SHA-1 が用いられている割合は 58.22%に上り、SHA-256 が用いられている割合も 33.32%と高い。

4.3　MD5 を利用しているアプリケーションについての追加分析

PlayDrone を利用して収集されたデータセットは 2 年前（2014 年 8 月）のデータであるため、現在では異なる証明書あるいは暗号アルゴリズムを用いている可能性がある。そこで PlayDrone データセットで APK の署名に MD5 を用いていた Android アプリ 52,286 個のうち、ランダムに選んだ 76 個（0.15%）のアプリを 2016 年 8 月 12 日に取得し、同様の分析を行った。取得には Nexus 7（2012）を用いた。76 個の標本により得られた統計情報は、MD5 の分散推定値 $0.043 = 0.00449^2$ を用いると、許容誤差 1%、信頼水準 95%を満たす。

結果は 2 つの APK が SHA-1 への移行を行い、それ以外は変わらず MD5 で署名を行っていた。母集団全体で統計的性質が同じであるという仮定を置けば、全体において同様の傾向があると言え、2 年前と変わらず多くの APK が MD5 のまま現在も利用されていることが考えられる。

表 9　PlayDrone データセットで APK 署名に MD5 を用いていたアプリに対する追加調査

署名アルゴリズム	個数
MD5	74
SHA-1	2

4.4　512 ビット RSA 鍵を利用しているアプリケーションについての追加分析

512 ビット RSA 鍵を利用しているアプリは 223 個見つかったが、そのうち 211 個に類似の特徴がみられた。これらのアプリで利用されている電子証明書の発行者と非発行者情報が

CN=Tasker User { 数字 } OU=Tasker Users O=Android Developers L=Anon ST=Anon C=US

という形式になっていた。数字はアプリごとに異なっているものの、それ以外の情報がすべて同じになっている。Tasker というキーワードは Android アプリや設定でユーザの固定動作を記憶させて自動的に動作させるツールとして知られているが、これらのアプリは Tasker をベースに作られた自動化ツールであることが予想される。本調査では Tasker による開発環境の提供等は発見できなかったものの、アプリ開発を容易にするサービスないし環境により鍵生成がされていて、そこにデフォルト（あるいはユーザが指定できず固定値として）弱い鍵を生成してしまうことが考えられる。

5.　署名偽造・悪用の脅威分析

5.1　512 ビット RSA 鍵の解読可能性

調査したうち、223 個の APK が 512 ビットの RSA 鍵を用いていることがわかった。512 ビットの RSA 鍵については、容易に素因数分解が可能であることが Valenta らにより示されている [14]。Valenta らによれば、Amazon EC2 を用いて 512 ビット RSA 鍵が 75 米ドルの費用で 4 時間かければ因数分解が可能としている。

安価に解読が可能であるため、これらのアプリケーションについては即座に対応することが必要である。秘密鍵（Private Key）が解読できてしまえば、再パッケージ化も容易であり、さらに保護レベルが signature または signatureOrSystem を持つパーミッションを持つアプリに対する悪用も自由に行えてしまう。

5.2　Chosen Prefix Collision Attack の可能性

MD5 は 2005 年に Wang らにより衝突を見つける方法 [15] が提案されて以来、数々の研究から MD5 の衝突に対する

脆弱性が確認されている。

2007 年に Stevens らによって提案された Chosen Prefix Collision Attack[16] は、MD5 の衝突を起こすデータの作成手法である。そこではメッセージペア (P_1,P_2) と調整用のデータペア (M_1,M_2) を利用して攻撃を行う。翌年の 2008 年に Sotirov らは実際に Chosen Prefix Collision Attack を用いて CA の証明書を偽造することに成功している [17]

Stevens らの Chosen Prefix Collision Attack を用いた証明書の偽造では、衝突させる 2 つのデータにおいて、衝突するための調整用データを入れていた。そのため、すでに用意されているデータに対しての適用は困難であった。Sotirov らの偽造においても、CA の証明書偽造に用いるために必要な正規発行 Web サイト証明書には、あらかじめ衝突させるためのブロックを含ませておく必要があった。APK を MD5 で署名しているデータに対する Chosen Prefix Collision Attack を考えた場合、すでに署名済みであり攻撃者が署名前の時点で衝突のためのブロックを含ませておくことが困難であるため、調整用のデータペアのうち片方が固定されることとなる。このことは、APK における MD5 偽造の難しさを示している。

一方で、APK への電子署名特有の攻撃容易性もある。たとえば、MD5 を用いている開発者が複数のアプリを提供している場合、攻撃者が衝突対象とするハッシュ値は複数得られることとなり、調整用データペアの柔軟性が増す。さらに、APK への署名では、SF ファイルに含まれる各ファイルのハッシュ値は、その前のファイルのハッシュ値の内部状態を保持したまま計算がされる。これにより、それぞれのファイルの一意性（完全性）の保証のみならずファイルの順序の保証も狙っているものと考えられるが、これは Chosen Prefix Collision Attack における IHV そのものということができる。そして、悪意のあるユーザは容易に任意のデータファイルを加えることができる。たとえば画像形式などのリソースファイルを用意しておき、実際にはアプリ内で利用しないことが可能である。これにより自由に IHV を調整することができるようになるため、調整用のデータの片方についての自由度は非常に高いといえる。

5.3 パーミッション悪用による脅威の可能性

パーミッションの保護レベルが signature または signatureOrSystem のものは、そのパーミッションを定義しておりかつ同じ署名の APK であれば、そのパーミッションを利用することができる。

APK の署名に MD5 を用いているアプリの中で保護レベルに signature が設定されているパーミッションが利用されているか追加調査を行った。調査結果を表 10 に示す。

このうち、多く利用されている「BIND_DEVICE_ADMIN」は、アプリで DeviceAdministration を利用するために用いる。DeviceAdministration は Android 2.2 から導入され、パスワードの長さの設定や、端末のリモートワイプなどの端末管理に関わる機能を可能にする。この機能は大きく分けて DeviceAdminReceiver、DevicePolicyManager、DeviceAdminInfo の 3 つのクラスで構成されている。この中の DeviceAdminReceiver クラスは、ユーザーが行ったセキュリティ設定に関する変更通知をシステムから受けるためのクラスであり、「BIND_DEVICE_ADMIN」は前述のクラスで作成した Receiver の permission 属性に宣言し、Manifest ファイルに登録する。

「BIND_INPUT_METHOD」は、アプリで IME（Input Method Editor）を利用するために用いる。IME は文字入力補助のソフトウェアで、これを利用してソフトウェアキーボードアプリの作成などを行う。IME を実装するには InputMethodService を継承したクラスが必要であり、BIND_INPUT_METHOD はこのクラスの permission 属性に宣言する。service は expoted の値を true にしたり、上記のような保護レベルに signature を持つパーミッションを設定した場合は、他のアプリでもその service を利用す

表 10 保護レベルに signature を含むパーミッションと利用していたアプリ数

パーミッション名	アプリ数
BIND_ACCESSIBILITY_SERVICE	15
BIND_CARRIER_SERVICES	0
BIND_CHOOSER_TARGET_SERVICE	0
BIND_CONDITION_PROVIDER_SERVICE	0
BIND_DEVICE_ADMIN	100
BIND_DREAM_SERVICE	1
BIND_INCALL_SERVICE	0
BIND_INPUT_METHOD	48
BIND_MIDI_DEVICE_SERVICE	0
BIND_NFC_SERVICE	7
BIND_NOTIFICATION_LISTENER_SERVICE	3
BIND_PRINT_SERVICE	1
BIND_QUICK_SETTINGS_TILE	0
BIND_SCREENING_SERVICE	0
BIND_TELECOM_CONNECTION_SERVICE	0
BIND_TEXT_SERVICE	6
BIND_TV_INPUT	0
BIND_VOICE_INTERACTION	0
BIND_VPN_SERVICE	16
BIND_VR_LISTENER_SERVICE	0
BIND_WALLPAPER	357
CLEAR_APP_CACHE	472
MANAGE_DOCUMENTS	55
READ_VOICEMAIL	0
SYSTEM_ALERT_WINDOW	1520
WRITE_SETTINGS	1288
WRITE_VOICEMAIL	0
製作者が定義した Permission	3112

ることができる。利用方法によっては、署名を偽造した悪意のあるアプリがこのパーミッションを悪用してユーザの入力情報を取得する可能性が考えられる。

製作者が定義するパーミッションで最も多かったものが C2D_MESSAGE であった。これは製作者が定義した Permission としてはプッシュ通知を受け取るため、G2DM を利用するときに「{ パッケージ名 }.permission.C2D_MESSAGE」として独自定義をしているものである。署名が同じアプリからはプッシュ通知の内容が漏えいしてしまう可能性が考えられる。

また製作者が定義するパーミッションの一つに、ContentsProvider がある。通常、アプリが管理する SQLite などのデータは他アプリからアクセスできないが、ContentsProvider を利用すると共有が可能になる。ContentsProvider はパーミッションの保護レベルでアクセス範囲の設定ができ、signature を含む値に設定すると同じ署名を持つアプリがその ContentsProvider のデータにアクセスできるようになる。これを悪用すると情報漏えいの恐れがある。

5.4 sharedUserID 悪用による脅威の可能性

Android アプリは通常、それぞれのプロセスを持ってそのプロセス内でリソースの参照や、処理を行う。他のアプリが許可なく別のアプリのリソースやデータや処理を得ることができない。しかし、AndroidManifest.xml で sharedUserID を設定し、その ID を宣言した同じ証明書をもつアプリ同士は、同じプロセス内に属することができ、データへのアクセスや処理を共有することができる。sharedUserID はデフォルトでは設定されていないが、設定した sharedUserID は AndroidManifest.xml から確認することができる。APK への署名に MD5 が用いられているアプリ群に対して sharedUserID の利用を調査したところ、430 個のアプリにおいて sharedUserID の利用が確認された。署名を偽造した悪意のアプリにより sharedUserID を悪用した情報漏えい等の脅威があると言えよう。

6. まとめ

Android アプリに付与される電子署名は、データの完全性保証が目的で付与されていると考えられる。しかし完全性保証だけでなく、同一開発者によるアプリ間の連携にも利用されている。本研究の調査では、Google Play マーケット上の 117 万のアプリにおいて、多くの APK が脆弱ないし強くない暗号鍵が用いられ、また脆弱ないし強くないハッシュ関数が用いられていることが分かった。これらの結果が示すことは、完全性保証が崩れる恐れだけでなく、暗号鍵の解読やハッシュ値の偽造により悪意を持って意図しないアプリ間連携がされる可能性である。アプリ間連携に署名を用いることは、現状の暗号利用ではリスクがあるため、適切な暗号利用が求められる。暗号技術に十分な知識のない開発者が適切な暗号を利用することは困難と考えられることもできるため、開発者による自発的な対策のみならず開発環境やマーケット審査の面でも暗号利用に注意をし適切な暗号利用を勧めることが重要であると考える。

謝辞

本研究の一部は科研費（16H02832）の助成を受けたものである。

また公約数分析に関して助言頂いた NTT セキュアプラットフォーム研究所の渡邉卓弥氏に感謝をいたします。

参考文献

[1] T. Watanabe, M. Akiyama, T. Sakai, H. Washizaki, T. Mori, "Understanding the Inconsistencies between Text Descriptions and the Use of Privacy-sensitive Resources of Mobile Apps", SOUPS2015, 2015

[2] C. Ren, Y. Zhang, H. Xue, T. Wei, P. Liu, "Towards Discovering and Understanding Task Hijacking in Android", The 24th USENIX Security Symposium, 2015

[3] N. Heninger, Z. Durumeric, E. Wustrow, J. A. Halderman, "Mining Your Ps and Qs: Detection of Widespread Weak Keys in Network Devices", 21st USENIX Security Symposium, 2012

[4] P. Svenda, M. Nemec, P. Sekan, R. Kvasnovsky, D. Formanek, D. Komarek, V. Matyas, "The Million-Key Question Investigating the Origins of RSA Public Keys", The 25th USENIX Security Symposium, 2016

[5] CERT/CC, "Multiple Android applications fail to properly validate SSL certificates", `http://www.kb.cert.org/vuls/id/582497`, 2014

[6] M. Weir, S. Aggarwal, M. Collins, and H. Stern. Testing metrics for password creation policies by attacking large sets of revealed passwords. In Proceedings of the 17th ACM Conference on Computer and Communications Security, CCS '10, pages 162-175, New York, NY, USA, 2010. ACM.

[7] M. Dell'Amico and M. Filippone. Monte carlo strength evaluation: Fast and reliable password checking. In Proceedings of the 22Nd ACM SIGSAC Conference on Computer and Communications Security, CCS '15, pages 158-169, New York, NY, USA, 2015. ACM.

[8] P. G. Kelley, S. Komanduri, M. L. Mazurek, R. Shay, T. Vidas, L. Bauer, N. Christin, L. F. Cranor, and J. Lopez. Guess again (and again and again): Measuring password strength by simulating password-cracking algorithms. In Security and Privacy (SP), 2012 IEEE Symposium on, pages 523-537, May 2012.

[9] B. Ur, P. G. Kelley, S. Komanduri, J. Lee, M. Maass, M. L. Mazurek, T. Passaro, R. Shay, T. Vidas, L. Bauer, N. Christin, and L. F. Cranor. How does your password measure up? the effect of strength meters on password creation. In Proceedings of the 21st USENIX Conference on Security Symposium, Security'12, pages 5-5, Berkeley, CA, USA, 2012. USENIX Association.

[10] B. Ur, S. M. Segreti, L. Bauer, N. Christin, L. F. Cranor, S. Komanduri, D. Kurilova, M. L. Mazurek, W. Melicher, and R. Shay. Measuring real-world accuracies and biases in modeling password guessability. In 24th USENIX Security Symposium (USENIX Security 15), pages 463-481,

Washington, D.C., Aug. 2015. USENIX Association.

[11] Android Developers , "Sign Your App", https://developer.android.com/studio/publish/app-signing.html, 2016/08/12 アクセス https://developer.android.com/studio/publish/app-signing.html

[12] Android Developers , "アプリケーションへの署名", https://developer.android.com/guide/publishing/app-signing.html, 2016/08/12 アクセス

[13] N. Viennot, E. Garcia, J. Nieh, "A Measurement Study of Google Play", ACM SIGMETRICS 2014, 201

[14] L. Valenta, S. Cohney, A. Liao, J. Fried, S. Bodduluri, N. Heninger, "Factoring as a Service", FC'16, 2016

[15] X. Wang, H. Yu, "How to Break MD5 and Other Hash Functions", EUROCRYPT 2005, 2005

[16] M. Stevens, A. Lenstra, B. Weger, "Chosen-Prefix Collisions for MD5 and Colliding X.509 Certificates for Different Identities", EUROCRYPT 2007, 2007

[17] A. Sotirov, M. Stevens,J. Appelbaum, A. Lenstra, D. Molnar, D. A. Osvik, B. Weger, "MD5 considered harmful today Creating a rogue CA certificate", 25th Chaos Communication Congress, 2008