東邦大学

生命（いのち）の科学で未来をつなぐ

Androidのコード署名に学ぶ IoT時代のソフトウェアアップデート

金岡 晃（東邦大学）

2018年4月17日

Agenda

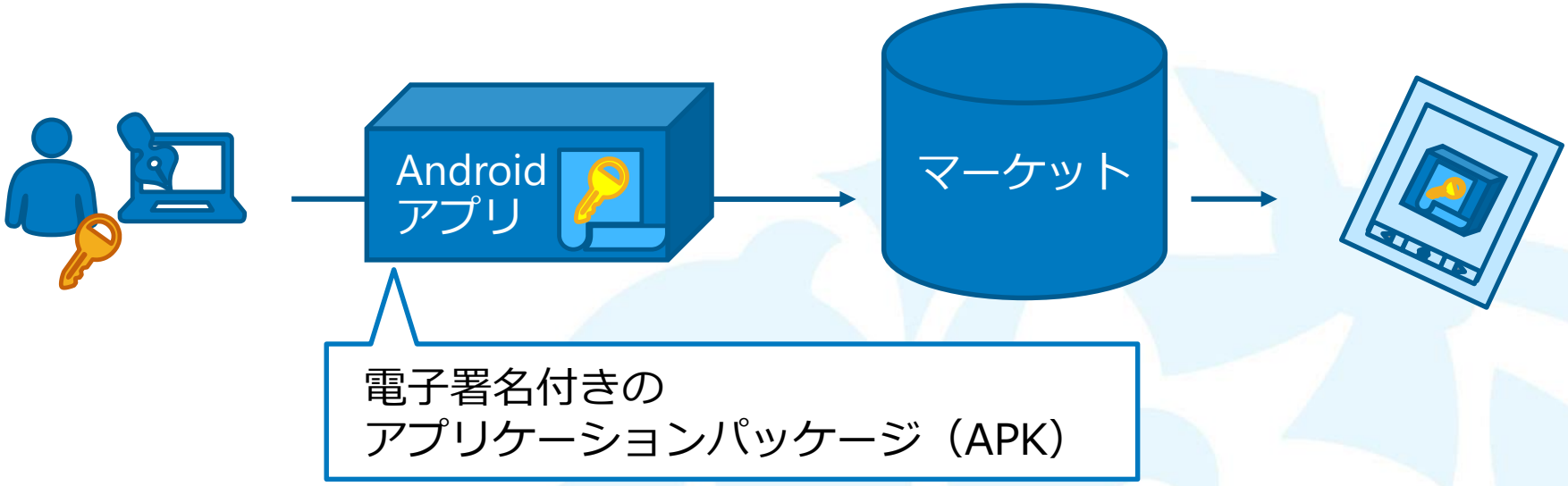
- Androidアプリへのコード署名の仕組み
- Androidアプリへのコード署名付与の現状
- 弱い署名が施される原因調査
- あるべき姿の考察

Androidアプリケーションのアップデート

コード署名を利用

インストールされるアプリケーションパッケージ（APK）が、すでにインストールされているAPKと比較して
・同じパッケージ名
・同じ署名者による署名
・バージョン番号が新しい
場合はアップデートとみなされる

Androidアプリケーションと電子署名



Google Playにおける電子署名に関連する要件と推奨

- 電子署名に用いる公開鍵暗号アルゴリズム：RSA、DSA
- 証明書有効期限：2033年10月22日以降を必須、10000日以上の有効期間を推奨

“アプリケーションへの署名, AndroidDevelopers”,
https://developer.android.com/guide/publishing/app-signing.html?hl=ja

APKの構造と電子署名(RSA鍵の場合)

- META-INF
  - MANIFEST.MF :ファイル名とそのハッシュ値が含まれたテキストファイル
  - CERT.SF :ファイル名とそのハッシュ値が含まれたテキストファイル
  - CERT.RSA :署名のハッシュ値と証明書（PKCS#7の署名データ）
- lib
- res
- assets
- AndroidManifest.xml
- classes.dex
- resources.arsc

「CERT.SF」を署名鍵で署名して、生成する

Androidアプリケーションでの電子署名の用途

電子署名の意味

証明書

信頼された認証局からの発行不要
＝自己署名証明書の利用が可能

署名者の同一性確認

データの完全性の確認

PKI技術は使うが、PKIが提供する信頼機構は使わない

電子署名の応用

署名者の同一性確認により以下のことが可能

- アプリのアップグレード
- アプリのモジュール性実現
- コードとデータの共有

アプリケーションのアップグレード
App upgrade

https://developer.android.com/studio/publish/app-signing.html

“アプリケーションのアップグレード - アプリケーションのアップグレードをリリースするとき、ユーザーが新バージョンにシームレスにアップグレードできるように、アップグレードされたアプリケーションにも同じ証明書で署名します。システムがアプリケーションのアップデートをインストールする際は、新バージョンの証明書のいずれかが旧バージョンの証明書と一致する場合、システムがアップデートを許可します。一致する証明書を使用せずに署名する場合は、アプリケーションに別のパッケージ名を割り当てる必要があります。この場合、新しいバージョンがまったく新しいアプリケーションとしてインストールされます。”

“When the system is installing an update to an app, it compares the certificate(s) in the new version with those in the existing version. The system allows the update if the certificates match. If you sign the new version with a different certificate, you must assign a different package name to the app—in this case, the user installs the new version as a completely new app.”

アプリケーションのモジュール性
App modularity

https://developer.android.com/studio/publish/app-signing.html

“Android システムでは、アプリケーションが要求する場合、同じ証明書で署名されたアプリケーションを同じプロセスで実行できます。これにより、システムはこれらを単一のアプリケーションとして取り扱います。このようにすればアプリケーションをモジュールとして配備でき、ユーザーは必要に応じて各モジュールを個別に更新できます。”

“Android allows APKs signed by the same certificate to run in the same process, if the apps so request, so that the system treats them as a single app. In this way you can deploy your app in modules, and users can update each of the modules independently.”

許可によるコード/データ共有
Code/data sharing through permissions

https://developer.android.com/studio/publish/app-signing.html

“Android システムでは、署名ベースの権限付与を実施しているため、アプリケーションは指定された証明書で署名されている別のアプリケーションに機能を提供できます。同じ証明書で複数のアプリケーションに署名し、署名に基づいた権限のチェックを行うことで、アプリケーションはコードとデータを安全な方法で共有できます。*”*

“Android provides signature-based permissions enforcement, so that an app can expose functionality to another app that is signed with a specified certificate. By signing multiple APKs with the same certificate and using signature-based permissions checks, your apps can share code and data in a secure manner.”

電子署名悪用による脅威

同一署名者へのなりすまし

以下の3つが可能に

- アプリのアップグレード
- アプリのモジュール性実現
- コードとデータの共有

→

アップグレードによる上書き

同一モジュール悪用

パーミッション悪用

なりすましの実現

- 鍵の漏えい
- 検証鍵の解析による署名鍵回復
- 署名付きアプリと同じハッシュ値となるデータの生成

APK電子署名の現状：Fahlらの調査

S. Fahl, S. Dechand, H. Perl, F. Fischer, J. Smrcek, and M. Smith,
“Hey, NSA: Stay Away from my Market! Future Proofing App Markets against Powerful Attackers”,
ACM CCS 2014

分析対象

- 対象：Google Play上の無料アプリ
- APK数：989,935
- 取得時期：2014年4月

分析結果

自己署名証明書	99.98% (380,285/380,345)
署名アルゴリズム	2860個がMD5、23個がMD2
鍵サイズ	51.6%がRSA2048、48.15%がRSA1024、0.07%がRSA512（277）
有効期限	70%が推奨期限内、12.3%が100-1000年、2033が1000年以上

APK電子署名の現状：吉田らの調査@CSS2016

吉田 奏絵、今井 宏謙、芹沢 奈々、森 達哉、金岡 晃、
“Androidアプリケーションにおける電子署名の大規模調査”，CSS2016

分析対象

- 対象：PlayDroneデータセット（Google Play上の無料アプリ）
- APK数：1,177,599
- 取得時期：2014年4月

分析結果

自己署名証明書	99.92% (1,177,185/1,177,599)
署名アルゴリズム	・証明書の署名：1.56%MD5（18,380） ・APKの署名：4.49%がMD5（52,866）
鍵サイズ	49.24%が2048、48.13%が1024、 0.02%がRSA512ビット鍵（223）
有効期限	25年以上が80.58%（949,193）

署名アルゴリズムと鍵サイズ

Algorithm and Key Size	# of APKs	
RSA512	223	0.02%
RSA1024	567,055	48.13%
RSA2048	580,134	49.24%
RSA4096	3,077	0.26%
DSA1024	26,697	2.27%
Others	929	0.08%

512ビット鍵はAmazon EC2を使って75ドルで4時間で解読できる（Valenta@FC2016）

Algorithm	# of Certs	
MD5	52,866	4.49%
SHA-1	1,123,779	95.45%
SHA-256	680	0.06%

衝突攻撃は容易

衝突が発見（2017）

APK電子署名の現状：吉田らの調査@SCIS2018

吉田 奏絵、今井 宏謙、芹沢 奈々、森 達哉、金岡 晃、
“Androidアプリケーションに対する弱い電子署名の脅威分析”, SCIS 2018

弱いアルゴリズム利用のリスク明確化

- 他のデータセットでの調査
- ダウンロード数調査
- 2016年での状況調査

弱いアルゴリズム利用の原因分析

- 開発環境の初期設定調査
- アプリ生成アプリ
- アプリ作成補助サービス

対策ガイドラインの検討

- OS開発者
- アプリ開発者
- マーケット管理者
- アプリ作成補助サービス事業者

他のデータセットを含めた調査

データセット	APK数	署名数	証明書数
PlayDrone	1,177,599	1,178,118	1,178,116
Alandroid	10,655	10,657	10,654
APPVN	34,415	34,433	34,485
Aptoide	138,122	138,172	138,291
Baidu	138,004	138,089	138,122
Blackmart	100,156	100,213	100,367
CafeBazaar	54,034	54,069	54,109
entumovil	235	235	235
GetJar	37,715	37,682	37,689
Mobogenie	31,546	31,570	31,590
MoboMarket	20,095	20,102	20,140
Uptodown	59,428	59,434	59,557
Yandex	22,964	22,969	22,975
zhushou360	204,416	204,567	204,673
Androzoo	3,834,514	3,830,320	3,839,527
Total	**5,863,898**	**5,867,415**	**5,870,531**

自己署名証明書

99.85% (5,862,531)

署名アルゴリズム

APKの署名：5.96%MD5 (349,478)

鍵サイズ

47.74%が1024、0.01%がRSA512 (837)

有効期限

25年以上が97.68% (5,734,146)

他のデータセットを含めた調査

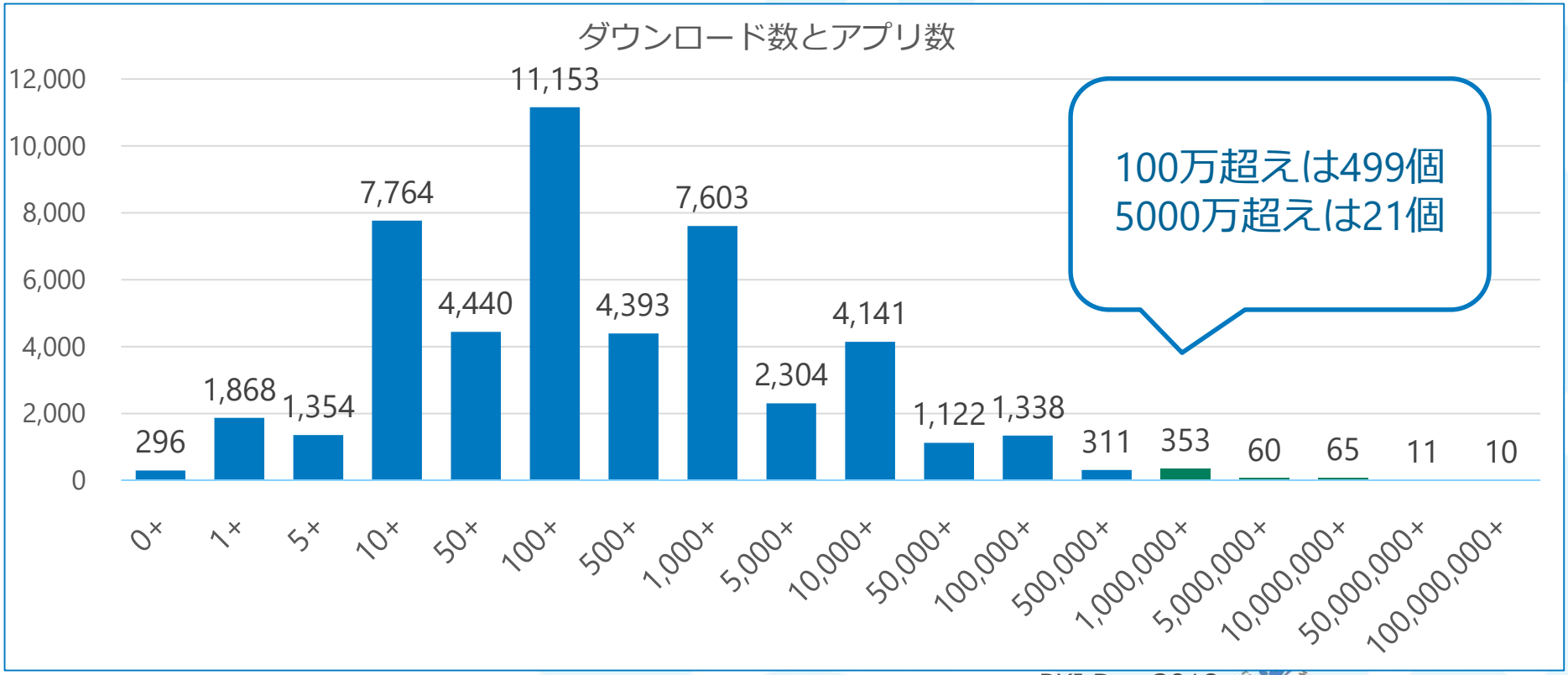
調査項目		Fahlら@SEC2014	吉田ら@CSS2016	本研究
自己署名証明書		99.98%	99.92%	99.85%
MD5利用(*1)		0.29%	4.49%	5.96%
鍵サイズ	RSA2048	51.6%	49.24%	49.80%
	RSA1024	48.15%	48.13%	47.74%
	RSA512	0.07%	0.02%	0.01%
有効期限		70%が推奨期限内	80.58%が25年以上	97.68%が25年以上

*1: Fahlらの結果は、証明書の署名アルゴリズムとAPKの署名アルゴリズムのどちらについての調査かの明言がない

弱いアルゴリズム利用のリスク明確化

ダウンロード数

PlayDroneデータセット内のAPK署名にMD5を利用していた52,866個のAPKのうちMetadata（周辺情報）があった48,586個のAPKに対してダウンロード数の調査を行った



弱いアルゴリズム利用のリスク明確化

2016年12月での状況調査

- PlayDroneデータセット内でMD5署名利用と512ビットRSA鍵利用をしていたアプリを再度ダウンロード
- Nexus7（2012）

アルゴリズム	アプリ数	割合
MD5	24,244	89.52%
SHA-1	2,801	10.34%
SHA-256	37	0.14%

MD5利用

- 52,866個中22,517個のAPK取得
- 89.52%が引き続きMD5を利用

512ビットRSA鍵利用

- 223個中78個のAPK再取得
- すべてが引き続き512ビットRSA鍵を利用

電子署名悪用による脅威

同一署名者へのなりすまし

以下の3つが可能に

- アプリのアップグレード
- アプリのモジュール性実現
- コードとデータの共有

→

- アップグレードによる上書き
- 同一モジュール悪用
- パーミッション悪用

なりすましの実現

- 鍵の漏えい
- 検証鍵の解析による署名鍵回復
- 署名付きアプリと同じハッシュ値となるデータの生成

→ **無視できるリスクではない**

原因分析：開発環境の初期設定調査

APKへの代表的な署名方法

- KeytoolとJarsignerを利用してコマンドラインで署名

→ JavaSE（JDK）に依存

- EclipseのADTプラグインでADT Export Wizard利用

→ JDKとAndroid Development Tools（ADT）に依存

- Android StudioでGenerate Signed APKを利用

→ Android Studioに依存

Java SE

MD5署名や512ビットRSA鍵の主原因とは考えにくい

Java SDK	Key Alg.	Key Size	Sign for APK	
version 6	RSA	1024	MD5withRSA	MD5withRSA
	DSA	1024	SHA1withDSA	SHA1withDSA
version 7	RSA	2048	SHA256withRSA	SHA256withRSA
	DSA	1024	SHA1withDSA	SHA256withRSA
version 8	RSA	2048	SHA256withRSA	SHA256withRSA
	DSA	1024	SHA1withRSA	SHA1withRSA

Eclipse with ADT

ADT ver.	JDK	Key Alg.	Key Size	Sign for APK	Sign for Cert
22.6.3.v2014	6.0	RSA	1024	SHA1	SHA1withRSA
23.0.7.2120684	8.0	RSA	2048	SHA1	SHA256withRSA

Android Studio

Studio ver.	JDK	Key Alg.	Key Size	Sign for APK	Sign for Cert
1.0	8.0	RSA	2048	SHA1	SHA256withRSA
2.1.3	8.0	RSA	2048	SHA1	SHA256withRSA

原因分析：512ビットRSA鍵

類似傾向の発見

PlayDroneデータセット上で512ビットRSA鍵を使っていたアプリ
211 / 223 個のAPKの証明書DNに類似傾向

CN= <ある固定キーワード> {アプリごとの数値列} OU= <ある固定キーワード> O=Android Developers L=Anon ST=Anon C=US

調査の結果、特定のアプリにより生成されたアプリであることが判明

2016年の段階でも当該アプリは存在し、
引き続き生成されたアプリは512ビットRSA鍵で署名されていたことを確認

原因分析：アプリ作成補助サービス

- アプリ開発の補助を行う特定のツールやサービス
- APK署名にMD5を利用しているアプリの証明書やパッケージ名に共通する文字列を分析

開発ツール名	個数	割合
ツール・サービスA	9,515	18.00%
ツール・サービスB	1,428	2.70%
ツール・サービスC	346	0.65%
ツール・サービスD	278	0.53%
ツール・サービスE	224	0.42%
ツール・サービスF	232	0.44%
ツール・サービスG	178	0.34%

まとめ

要因の考察

スマートフォン端末と
アプリの拡大

幅広い開発者の参入

OSが提供する
アプリ機能の充実化

多機能性の一端を
電子署名による
保証を利用して実現

OS・アプリ構造の
複雑化

複雑化した構造、
脅威・リスク
の理解の困難性

現状

ミス・悪用による
不適切利用リスク
拡大

脅威・リスクと
電子署名・証明書・鍵への要件
のバランスが取れていない

アプリ機能と署名要件の乖離が
当初より拡大してきた可能性

あるべき姿の考察

コード署名による利便性

- アップデート
- モジュール化
- コード/データ共有

⟺

署名に関連する要件の不適切さ

- 鍵更新を許容しない モデル
- 脆弱なアルゴリズムが 利用可能
- 長期間の有効期限を持つ 証明書の強制

署名関連の要件を適切に

開発ツール側でも対応可能に

開発ツール側でも対応可能に

「現行PKIを使いこなせない開発者は、将来的な暗号技術も使いこなせない」という前提に立つべき

↓

開発者が意識しなくても（あるいは意識が少なくても）
安全に開発可能な開発ツールや環境、プラットフォームを構築

まとめ

Androidアプリへのコード署名の仕組み

Androidアプリへのコード署名付与の現状

- RSA/DSA鍵
- 証明書有効期限25年以上推奨
- 弱いアルゴリズム利用：RSA512、MD5

弱い署名が施される原因調査

- アプリやサービスにより簡易に生成されたアプリ

あるべき姿の考察

- 鍵と証明書が更新可能なエコシステム
- 開発者フレンドリーな環境