Computer Security Symposium 2019
21 - 24 October 2019

Androidアプリケーションにおける暗号化API利用に関する静的解析手法の考察

角田 大輔[1,a)]　松浦 幹太[2,b)]

概要：近年，デジタル・フォレンジックと呼ばれる電磁的記録に対する科学的調査技術の重要性が高まっている．特にスマートフォンに関しては，比較的短期間で急速に普及したこともあってフォレンジックの対象となる機会が非常に多く，スマートフォンに対するフォレンジック技術の向上が望まれている．
スマートフォンについては従来よりアプリケーションのセキュリティ向上に関する様々な研究がなされている．デジタル・フォレンジックに関する研究は従来のアプリケーションセキュリティに関する研究と比較するとまだ少ないものの，最近では従来のセキュリティを目的としたAndroidアプリの解析手法に関する研究をデジタル・フォレンジックの研究に応用する動きがある．
デジタル・フォレンジックにおける課題の1つには電磁的記録として保存されるデータに暗号化処理が施されてる場合への対処があり，そのような場合においては復号方法の解明が重要となるケースがある．本稿では，セキュリティ診断を目的として作られている既存のAndroidアプリの静的解析フレームワークを利用し，Androidアプリにおける暗号化APIの利用状況を解析する手法の検討を行った．

キーワード：Android，静的解析，暗号化API，デジタル・フォレンジック

A Study on Analysis Methods of Crypto API Usages on Android Apps using a Static Analysis Framework

Daisuke Sumita[1,a)]　Kanta Matsuura[2,b)]

Abstract: Recently, digital forensics, methods of scientific investigation for digital devices, become more important. Because of a rapid growth in smartphone popularity, we encounter the challenges of developing smartphone forensic techniques.
There are various smartphone security researches concerning improving application security. While less researchs are about digital forensics, however, Some recent researches leverage existing security researches of Android apps for digital forensics.
Data encryption is one of the major challenges of digital forensics. In such cases, the development of decryption method is essential. In this paper, we study analysis methods of crypto API usages on Android apps, using an existing static analysis framework for security vetting of Android apps.

Keywords: Android, Static Analysis, Crypto API, Digital Forensics

1. はじめに

パソコンやスマートフォンに代表される情報通信機器が広く一般に普及し，そのような情報通信機器が生活に不可欠なものとなっている．特にスマートフォンについては近年爆発的に普及し，世帯保有率は約8割に達しパソコンを

[1] 警察庁
National Police Agency, Japan
[2] 東京大学生産技術研究所
Institute of Industrial Science, The University of Tokyo
[a)] sumita@post.cyberpolice.go.jp
[b)] kanta@iis.u-tokyo.ac.jp

上回る状況となっている [1].

これら情報通信機器を含む電子機器等に保存される情報は，犯罪捜査等において重要な客観証拠になる場合がある．そのため，従前よりデジタル・フォレンジックと呼ばれる電磁的記録に対する科学的調査技術が重要視されてきたが，特に近年ではスマートフォンに対するデジタル・フォレンジックの必要性が非常に高まっている．

これまでのスマートフォンに関する研究としては，アプリケーションのセキュリティを保つという観点から，特に Android アプリケーションの解析手法に関する研究が盛んであり [2], [3]，それらの中には研究の成果として解析に使用するプログラムを広く一般に公開しているものもある [4], [5]．また，最近ではこのようなプログラムを利用し，Android アプリケーションの解析を行って重要な情報が保存される場所を明らかにすることで，デジタル・フォレンジックに活用する研究も現われている [6], [7]．これらの研究においては情報の保存場所，すなわちファイルパスを特定することを課題とし，Android アプリケーションの解析技術を活用している．

このように情報の保存場所を特定することはデジタル・フォレンジックにおける大きな課題の 1 つであるが，別の課題として，データが暗号化されて保存される場合への対処があげられる．例え情報の保存場所が判明したとしても，それらの情報が容易に可視化・可読化できない形で保存されていればデジタル・フォレンジックの障壁となり得るからである．そのような課題に対して，これまでの研究では主に個別のアプリケーション毎に人手による調査を行い，各アプリケーションで使用されている暗号方式を明らかにすることで対処してきた [8], [9], [10].

本稿では，より一般的な対処として，アプリケーション内でどのように暗号化 API が利用されているかを調査する手法について検討する．これまでの研究で Android アプリケーションの暗号化 API の利用に着目したものとしては，利用の誤り，すなわち安全性の低い利用方法を選択して実装していないかを検知するもの [11], [12] があるが，デジタル・フォレンジックに活用するためには暗号化 API の具体的な利用状況を明らかにする必要がある．そのため，本稿では暗号化 API に関してより詳細な解析を行った利用状況の調査について，既存の静的解析フレームワークを活用した実験を行い，利用状況を明らかにできることを確認する．

本稿の構成は以下のとおりである．第 2 節で Android アプリケーションの基本的な事項について説明する．第 3 節で暗号化 API の解析手法を検討し．第 4 節で検討した解析手法の実験を行う．最後に，第 5 節で結論を述べる．

2. Android アプリケーション

本稿で解析の対象としている Android アプリケーションについて，本節でその基本的な構成を述べる．また，Android アプリケーションにおいて標準的に利用されている暗号化 API について説明する．

表 1　APK ファイルの構成
Table 1　APK File Structures

Component	Description
`AndroidManifest.xml`	アプリに関する重要な定義等を記録
`Classes.dex`	Dalvik 実行可能バイトコード
`lib/`	ネイティブバイナリを保存
`assets/`	アプリ内で使用するファイルを保存
`res/, resources.arsc`	アプリのリソースを保存
`META-INF/`	電子署名等を保存

2.1 アプリケーションの構成

Android OS において動作するアプリケーションは，APK(Application PacKage) というファイル形式で配布される．APK ファイルはアプリケーションとしての動作に必要なファイルが ZIP 形式で圧縮されており，その構成は表 1 のとおりである．

アプリケーションの動作そのものとなる実行ファイルは Dalvik 実行可能な `Classes.dex` 及び `lib/`フォルダに保存されるネイティブバイナリのファイルであるが，このうち本稿で解析の対象とするのは `Classes.dex` とする．

Android アプリケーションは Java 言語を使用して開発されるのが一般的であり，`Classes.dex` はアプリケーション開発者が Java 言語で記述したソースコードを Dalvik 実行形式にコンパイルしたもので，Android アプリケーションの動作の主要な部分を担うものとなる．

なお，Android アプリケーションの開発においてネイティブバイナリに暗号化機能を実装してアプリケーションで利用することは可能であるが，`Classes.dex` とネイティブバイナリでは解析手法が大きく異なり同様に扱うことは困難であるため，本稿ではネイティブバイナリに実装された暗号化機能については対象外としている．

2.2 標準的な暗号化 API

第 2.1 節で述べたとおり，本稿における解析の対象は Java 言語による記述が基となっている `Classes.dex` であるので，注目する暗号化 API についても Java 言語で標準的に利用できるの暗号化 API とする．

デジタル・フォレンジックにおいては各種の電磁的記録媒体に保存されている情報を取得・解析するが，対象とする媒体は不揮発性メモリであることが大多数である．不揮発性メモリに情報を記録する際に Android アプリケーションで標準的に用いられる暗号化 API としては `javax.crypto.Cipher` クラスが挙げられ [13]，実際に Google 社が Android アプリケーションの開発者向けに公開している標準的な暗号化の実装についても，`Cipher` クラスを使用した次に示すコード例が記載されている [14].

```
byte[] plaintext = ...;
KeyGenerator keygen = KeyGenerator.getInstance("AES");
keygen.init(256);
SecretKey key = keygen.generateKey();
Cipher cipher = Cipher.getInstance("AES/CBC/
    PKCS5PADDING");
cipher.init(Cipher.ENCRYPT_MODE, key);
byte[] ciphertext = cipher.doFinal(plaintext);
byte[] iv = cipher.getIV();
```

このコード例からも分かるとおり，暗号化の際には Cipher オブジェクトの初期化等で複数のパラメータを設定する必要があり，これらのパラメータを具体的な設定状況が暗号化の方式を決定する．そのため，本稿における暗号化 API の利用状況の調査としては，アプリケーションの解析を行ってこれらパラメータを得ることを目標とする．

3. 解析手法

本稿で解析の対象とする暗号化 API について，取得すべきパラメータを含めて具体的に列挙する．また，解析に利用する既存の静的解析フレームワークについても説明する．

3.1 解析対象とする暗号化 API

第 2.2 節で述べたとおり，Android アプリケーションで標準的な暗号化の実装には Cipher クラスが利用される．本稿では当該 API を対象とするが，特に注目するメソッドを初期段階の Cipher オブジェクトを生成する getInstance メソッド及び暗号化のための初期化処理を行う init メソッドとする．これらのメソッドの具体的な実行方法については表 2 に列挙するとおりであり，取得すべきパラメータについては下線で示す．

また，これら取得すべきパラメータに関して，パラメータ生成のために標準的に利用される API が存在する．それら API のうち本稿で解析対象とするものを具体的な実行方法を含めて表 3 に列挙する．これらの API では，実行時のパラメータ全てが暗号化方式の特定に必要なため，全てのパラメータを取得すべきものとする．

以上の表 2 及び表 3 に列挙した API について解析を行い，必要なパラメータを取得することを目指す．

3.2 解析に使用するプログラム

Android アプリケーションの解析に利用できるプログラムは複数存在するが，本稿では Wei らによって提案され [4]，オープンソースソフトウェアとして公開されている静的解析フレームワークである Argus-SAF[15] を利用することとした．Argus-SAF はコンポーネント間のデータフロー解析が効率的かつ高い正確性で行えるほか，Source と Sink を利用者が設定してテイント解析を行うことも可能であり，利用者が解析したい内容を自ら開発する場合にプラグインとして付け加えることも想定した設計であるため，本稿で設定した各種の暗号化 API のパラメータを取得するという問題もプラグインの形で実装することを目指す．

4. 実験

前節までで述べた解析対象とする暗号化 API について，それぞれの API を実装したアプリケーションを実際に作成し，作成したアプリケーションの解析により各 API に設定したパラメータが取得できることを確認する．

4.1 解析の対象とするアプリケーション

Android アプリケーションの解析手法の評価を目的として作られたベンチマークはすでに存在するが [16], [17]，本稿で対象としている暗号化 API が実装されているベンチマークは見付からなかったため，本稿では第 3.1 節で述べた各解析対象の暗号化 API をを実装した Android アプリケーションを作成することとした．

それぞれの暗号化 API を実装したアプリケーションの作成にあたっては，単純化のために次に示す条件のもと行った．

(1) API の呼び出し等一連の処理をを同一クラス内に実装した．
(2) 表 3 に列挙した各 API の実装においては，入力されるパラメータを全て固定値とした．
(3) 表 2 に列挙した各 API の実装については次のとおりとした．
- getInstance メソッド呼び出し時のパラメータは全て固定値とした．
- init メソッド呼び出し時のパラメータは先に固定値を使用して実装した (2) の結果をそのまま入力とした．

なお，作成したアプリにおける暗号化 API の実装例は次に示すとおりである．この例において，取得すべきパラメータは IvParameterSpec クラスのインスタンス生成で使用している iVec の内容であり，解析により文字列"ThisIsMyInitVect"が取得できればよいものとする．

```
String initialVector = "ThisIsMyInitVect";
byte[] iVec = initialVector.getBytes();
IvParameterSpec ivParamSpec = new IvParameterSpec(
    iVec);
```

4.2 解析の実施

第 4.1 節で作成したアプリケーションを解析するために，Argus-SAF のテイント解析機能の利用及びプラグインの作成を行った．Argus-SAF にはすでに暗号化 API の誤利用を検知するプラグインが備えられているが，当該プラグインは Egele らが提案する暗号化 API の利用方法 [11] の

表 2　対象とする暗号化 API のメソッド
Table 2　Target Crypto API Methods

Target API Method	Method Description

`Cipher getInstance(String)`
`Cipher getInstance(String, String)`
`Cipher getInstance(String, Provider)`
	Returns a Cipher object

`void init(int, Key)`
`void init(int, Key, SecureRandom)`
`void init(int, Key, AlgorithmParameters)`
`void init(int, Key, AlgorithmParameters, SecureRandom)`
`void init(int, Key, AlgorithmParameterSpec)`
`void init(int, Key, AlgorithmParameterSpec, SecureRandom)`
	Initialize the Cipher object

表 3　Cipher クラスのメソッドの引数を生成する API のメソッド
Table 3　API Methods that generate variables in Cipher class

API Class	API Method	Method Description
`javax.crypto.spec.SecretKeySpec`	`void <init>(byte[], String)` `void <init>(byte[], int, int, String)`	Key Definition

`javax.crypto.spec.PBEKeySpec`	`void <init>(char[])`
	`void <init>(char[], byte[], int)`
	`void <init>(char[], byte[], int, int)`
		Key Definition

`javax.crypto.spec.PBEParameterSpec`	`void <init>(byte[], int)` `void <init>(byte[], int, AlgorithmParameterSpec)`	Key Definition
`javax.crypto.spec.IvParameterSpec`	`void <init>(byte[])` `void <init>(byte[], int, int)`	IV Definition

うち，次に示す 2 つのみが実装されている．

(1) `Cipher` クラスのインスタンス生成に ECB モードを使用しない
(2) CBC モードの暗号化においてランダムでない IV を使用しない

このうち (1) の具体的な実装としては，`Cipher` クラスの `getInstance` メソッドのパラメータとして ECB モードを使用する文字列が入力されていないかを判定している．また，(2) の具体的な実装としては `IvParameterSpec` クラスのインスタンス生成において定数のパラメータが入力されていないかを判定している．

いずれも本稿の目指しているパラメータ自体の取得までには至っていないが，拡張することにより各種暗号化 API のパラメータ取得が可能であるため，既存の実装を拡張して解析を実施することとした．

4.2.1　Cipher クラス以外のメソッドの解析

`Cipher` クラス以外のメソッドについてはそれぞれ固定値がパラメータとして入力されるため，Argus-SAF に備えられている暗号化 API の誤利用を検知するプラグインを拡張し，パラメータ自体が取得できることを確認した．

4.2.2　Cipher クラスのメソッドの解析

`Cipher` クラスのメソッドのうち，`getInstance` メソッドについては第 4.2.1 節と同様に固定値がパラメータとして入力されるため，同様にパラメータ自体が取得できることを確認した．`init` メソッドについては単純に固定値がパラメータとして入力されるわけではないため，Source を表 3 に列挙したパラメータを生成する API，Sink を `Cipher` クラスの `init` メソッドと設定したテイント解析を利用し，`init` メソッドに入力されるパラメータが特定できることを確認した．なお，設定した具体的な Source 及び Sink は図 1 に示すとおりである．

以上の結果をすべて組み合わせることにより，第 4.1 節で作成したアプリケーションを解析して暗号方式の判明に必要なパラメータの取得ができることを確認した．

5.　まとめと課題

本稿では，Android アプリケーションにおいて暗号化 API が具体的にどのように利用されているのかを解析する手法について検討を行い，既存の静的解析のフレームワークを利用することで実際に利用状況が調査できることを確認した．本稿の結果を応用することにより，Android 端末の解析においてアプリケーションが作成したデータが暗号化されている場合に効率的に暗号化手法を判明させられる可能性がある．

本稿で確認を行ったのは各暗号化 API に入力されるパラメータが単純な固定値である場合のみであるため，今後，パラメータが様々な処理を経て API に入力される場合や，動的に生成される値が入力される場合等への対応を検討

```
Ljavax/crypto/spec/SecretKeySpec;.<init>:([BLjava/lang/String;)V CRYPTO_FUNCTION -> _SOURCE_
Ljavax/crypto/spec/SecretKeySpec;.<init>:([BIILjava/lang/String;)V CRYPTO_FUNCTION -> _SOURCE_
Ljavax/crypto/spec/PBEKeySpec;.<init>:([C)V CRYPTO_FUNCTION -> _SOURCE_
Ljavax/crypto/spec/PBEKeySpec;.<init>:([C[BI)V CRYPTO_FUNCTION -> _SOURCE_
Ljavax/crypto/spec/PBEKeySpec;.<init>:([C[BII)V CRYPTO_FUNCTION -> _SOURCE_
Ljavax/crypto/spec/PBEParameterSpec;.<init>:([BI)V CRYPTO_FUNCTION -> _SOURCE_
Ljavax/crypto/spec/PBEParameterSpec;.<init>:([BIILjava/security/spec/AlgorithmParameterSpec;)V CRYPTO_FUNCTION -> _SOURCE_
Ljavax/crypto/spec/IvParameterSpec;.<init>:([B)V CRYPTO_FUNCTION -> _SOURCE_
Ljavax/crypto/spec/IvParameterSpec;.<init>:([BII)V CRYPTO_FUNCTION -> _SOURCE_
Ljavax/crypto/Cipher;.init:(ILjava/security/Key;)V -> _SINK_ 2
Ljavax/crypto/Cipher;.init:(ILjava/security/Key;Ljava/security/SecureRandom;)V -> _SINK_ 2
Ljavax/crypto/Cipher;.init:(ILjava/security/Key;Ljava/security/AlgorithmParameters;)V -> _SINK_ 2|3
Ljavax/crypto/Cipher;.init:(ILjava/security/Key;Ljava/security/AlgorithmParameters;Ljava/security/SecureRandom;)V -> _SINK_ 2|3
Ljavax/crypto/Cipher;.init:(ILjava/security/Key;Ljava/security/spec/AlgorithmParameterSpec;)V -> _SINK_ 2|3
Ljavax/crypto/Cipher;.init:(ILjava/security/Key;Ljava/security/spec/AlgorithmParameterSpec;Ljava/security/SecureRandom;)V -> _SINK_ 2|3
```

図 1 テイント解析のために設定した Source 及び Sink

Fig. 1 Souce and Sink definition for taint analysis

し，本稿で述べた手法を拡張していく必要がある．また，既存研究である保存されるファイルのパスを特定する研究と組み合わせて，暗号化されたデータがどのように保存されるか特定することで，より効率的なデジタル・フォレンジックを行うことができる可能性がある．

参考文献

[1] 総務省. 通信利用動向調査. http://www.soumu.go.jp/johotsusintokei/statistics/statistics05.html.

[2] Bradley Reaves, Jasmine Bowers, Sigmund Albert Gorski III, Olabode Anise, Rahul Bobhate, Raymond Cho, Hiranava Das, Sharique Hussain, Hamza Karachiwala, Nolen Scaife, Byron Wright, Kevin Butler, William Enck, and Patrick Traynor. *Droid: Assessment and Evaluation of Android Application Analysis Tools. *ACM Comput. Surv.*, Vol. 49, No. 3, pp. 55:1–55:30, October 2016.

[3] Li Li, Tegawendé F. Bissyandé, Mike Papadakis, Siegfried Rasthofer, Alexandre Bartel, Damien Octeau, Jacques Klein, and Le Traon. Static analysis of android apps: A systematic literature review. *Information and Software Technology*, Vol. 88, pp. 67 – 95, 2017.

[4] Fengguo Wei, Sankardas Roy, Xinming Ou, and Robby. Amandroid: A Precise and General Inter-component Data Flow Analysis Framework for Security Vetting of Android Apps. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, CCS '14, pp. 1329–1341, New York, NY, USA, 2014. ACM.

[5] Steven Arzt, Siegfried Rasthofer, Christian Fritz, Eric Bodden, Alexandre Bartel, Jacques Klein, Yves Le Traon, Damien Octeau, and Patrick McDaniel. FlowDroid: Precise Context, Flow, Field, Object-sensitive and Lifecycle-aware Taint Analysis for Android Apps. *SIGPLAN Not.*, Vol. 49, No. 6, pp. 259–269, June 2014.

[6] Xiaodong Lin, Ting Chen, Tong Zhu, Kun Yang, and Fengguo Wei. Automated forensic analysis of mobile applications on Android devices. *Digital Investigation*, Vol. 26, pp. S59 – S66, 2018.

[7] Chris Chao-Chun Cheng, Chen Shi, Neil Zhenqiang Gong, and Yong Guan. EviHunter: Identifying Digital Evidence in the Permanent Storage of Android Devices via Static Analysis. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, CCS '18, pp. 1338–1350, New York, NY, USA, 2018. ACM.

[8] Cosimo Anglano. Forensic analysis of whats app messenger on Android smartphones. *Digital Investigation*, Vol. 11, No. 3, pp. 201–213, 2014.

[9] Cosimo Anglano, Massimo Canonico, and Marco Guazzone. Forensic analysis of the ChatSecure instant messaging application on android smartphones. *Digital Investigation*, Vol. 19, pp. 44–59, 2016.

[10] Songyang Wu, Yong Zhang, Xupeng Wang, Xiong Xiong, and Lin Du. Forensic analysis of WeChat on Android smartphones. *Digital Investigation*, Vol. 21, pp. 3–10, 2017.

[11] Manuel Egele, David Brumley, Yanick Fratantonio, and Christopher Kruegel. An empirical study of cryptographic misuse in android applications. In *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security - CCS '13*, pp. 73–84, New York, New York, USA, 2013. ACM Press.

[12] Shao Shuai, Dong Guowei, Guo Tao, Yang Tianchang, and Shi Chenjie. Modelling analysis and auto-detection of cryptographic misuse in android applications. *Proceedings - 2014 World Ubiquitous Science Congress: 2014 IEEE 12th International Conference on Dependable, Autonomic and Secure Computing, DASC 2014*, pp. 75–80, 2014.

[13] Google. API Reference |Google Developers. https://developer.android.com/reference.

[14] Google. Cryptography |Android Developers. https://developer.android.com/guide/topics/security/cryptography.

[15] Argus group. arguslab/Argus-SAF: Argus static analysis framework. https://github.com/arguslab/Argus-SAF.

[16] DroidBench. secure-software-engineering/DroidBench: A micro-benchmark suite to assess the stability of taint-analysis tools for Android. https://github.com/secure-software-engineering/DroidBench.

[17] ICCBench. fgwei/ICC-Bench: Benchmark apps for static analyzing inter-component data leakage problem of Android apps. https://github.com/fgwei/ICC-Bench.