# Androidアプリケーションにおける静的解析を使用した暗号化API利用の特定

角田 大輔[1,a)]　松浦 幹太[2,b)]

**概要**：デジタル・フォレンジック調査において，携帯デバイスを調査対象とすることは必要不可欠な作業である．特にスマートフォンに関しては様々なデータが保存され，その中には有用なデータが含まれている．しかしながら，最近の傾向としてアプリケーションによりデータが暗号化される場合が少なくない．そのようなデータの暗号化は迅速な調査の障害となり，スマートフォン・フォレンジックにおける大きな課題の 1 つである．
従来の研究の多くは特定の 1 つのアプリに対する手動解析で暗号化処理の特定を行っているが，そのような作業はアプリ解析の知識や経験が必要な手間のかかる作業である．保存データを暗号化するアプリは数多く存在するが，その中でも多くのアプリが標準的な暗号化 API を使用している．本稿では，既存の Android アプリの静的解析フレームワークを使用し，アプリにおいて標準的な暗号化 API がどのように利用されているかを特定するツールの開発を行った．本ツールでは自動的にアプリを解析することが可能であり，暗号化処理の特定を容易に行うことができる．その結果，アプリによって暗号化されたデータの解析を迅速に行うことが可能となる．

**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

# Identifying Crypto API Usages in Android Apps using a Static Analysis Framework

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

***Abstract:*** Forensic analysis of mobile devices is essential work for digital forensic investigators. While there are various data stored in smartphones, some of the data is encrypted by applications. Data encryption is one of the major issues of digital forensics, preventing investigators from analyzing the data quickly.
In this work, we develop a tool to automatically analyze crypto API usages in Android apps. There are many Android apps which encrypt their data in smartphones using standard crypto APIs. In such cases, we can identify the cryptographic algorithms and parameters via application analysis, which helps us to analyze encrypted data. Most existing studies focus on single app, and rely on manual analysis, which requires a certain amount of skill and knowledge about reverse engineering. For this reason, we develop our tool which can analyze apps automatically, therefore we can easily identify crypto API usages in new apps.

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

## 1. はじめに

スマートフォンはその多機能性や利便性から, 現在の人々の生活において多くの役割を果たす物となっている．特に, スマートフォンの代表的な機能の 1 つに様々なアプリ

[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)] d.sumita.yd.y8@npa.go.jp
[b)] kanta@iis.u-tokyo.ac.jp

ケーションを動作させられることが挙げられるが, それゆえに使用者の利用形態に合わせ, 多くのサードパーティー製アプリをインストールした状態で使用されていることが多い．これらのインストールされたアプリは使用者に関する情報を含むデータを扱ったり保存したりするため, 犯罪捜査等において重要な客観証拠になる場合があり, スマートフォンに対するデジタル・フォレンジックは必要不可欠となっている．

スマートフォンに対するデジタル・フォレンジックにおいては, 解析の対象となるスマートフォンにインストールされる様々なアプリがどのように動作するかを把握する必要がある．特に, アプリによってスマートフォンに保存されるデータがどのような種類のものなのか, どこに保存されるのか, どのようなデータ形式で保存されるのか, といったことは解析を行う上で非常に重要なことである．しかしながら, そのようなことを判明させるのは必ずしも容易ではなく, その上スマートフォン用のアプリは非常に膨大な数が存在するため, 解析者はアプリの動作の把握に多くの時間を必要としている．

アプリの動作を明らかにする研究は盛んに行われているが, 既存の研究では人手に頼った解析を行っているものが多く, そういった手法は解析そのものに時間がかかったり, 解析者に高度な解析能力が必要であったりする問題がある．そのような問題を解決するため, 最近では Android のアプリを自動的に解析して保存されるデータの種類や場所を特定するという研究が複数なされている [1], [2], [3], [4]．しかしながら，それらの研究においてはデータの保存形式までは特定することができない．

近年の人々のプライバシーに関する懸念の高まりにより，スマートフォンのアプリがデータを扱う際に暗号化の処理を組み込むのは非常に一般的となっているが，一方でデジタル・フォレンジックの観点からはそういったデータの暗号化が解析を阻む要因となりうる．既存の研究ではそのようなデータの暗号化に注目し，いわゆるセキュリティアプリと呼ばれるような Android のアプリにおけるデータの暗号化の方法を明らかにしているものがある [5], [6]．このような研究によって暗号化されたデータの解析が可能になっているが，当該研究についても解析を人手に頼っているという問題がある．自動的な Android アプリの暗号解析に関する既存の研究も複数存在はするが [7], [8], [9]，それらの研究の大半が，安全性が十分でない不適切な暗号化の処理の実装をしていないかといったことを自動解析で検知するというアプリ開発者の観点によるものであり，暗号化して保存されるデータの解析に活用するには不十分である．

そこで本稿では，Android アプリによってどのようにデータが暗号化されるかを，アプリの自動解析によって特定することを目的とする．Android アプリの自動解析については多くの既存研究の成果が利用可能であり，本稿ではその 1 つである静的解析フレームワークの ”Amandroid” [10] を活用し，アプリにおける暗号化処理の特定を行う．

Android アプリにおける暗号化処理の実装には様々な方法が考えられるが，本稿では最も標準的な実装方法である `Cipher` API を使用した実装に関して API の利用方法の特定を行うこととし，同時に `Cipher` API の利用に必要な鍵や初期化ベクトルといったパラメータ生成のための API についても利用方法の特定を行う．また，これらの API の引数の導出過程に一般的に用いられやすい Android の API を選び，暗号化されたデータの復号に活用できるよう分類を行う．以上の暗号化に関する API 及び引数の導出に関する API の利用が特定できるように ”Amandroid” の拡張を行う．

本稿では拡張した解析フレームワークを使用し，実在する 139 個の Android アプリを解析して評価実験を行った．結果として復号処理に使用されている `Cipher` API を 200 箇所発見し，それに関係する鍵生成等の暗号化 API の使用を合計 178 箇所特定した．さらに，これらの暗号化 API に必要な引数の導出に使用された API を合計 212 箇所で特定した．

まとめると，本稿では次に示す結果が得られた．

- Android の標準的な暗号化 API の利用に際し，一般的に利用されることが多い API を抽出してその分類を行った．
- 既存の解析フレームワークを拡張し，Android アプリの暗号化 API 利用の特定を行うツールを開発した．
- 実在する 139 個の Android アプリに対して評価実験を行い，合計で 378 箇所の暗号化 API の使用を特定し，引数の導出に使用された API を合計 212 箇所で特定した．

本稿の構成は以下のとおりである．第 2 節で Android アプリケーションにおける暗号化処理の基本的な事項について説明し，第 3 節で暗号化 API の解析手法及びその実装について説明する．第 4 節で解析の実験を行いその結果について述べる．第 5 節で本研究の議論及び関連研究の紹介を行い，最後に第 6 節で結論を述べる．

## 2. 背景

第 1 節でも述べたとおり，スマートフォンではプライバシー情報を扱うことが多く，プライバシーに対する懸念に対応するために取り扱うデータに暗号化処理を施す場合がある．Android に関しては暗号化処理に利用できるライブラリ等が複数存在し，アプリで暗号化処理を実装する際にはそのようなライブラリ等を利用することが一般的である [11]．

その中でも最も一般的と考えられる標準状態の Android OS で使用できる暗号化 API について説明し，その後それらの暗号化 API に対して解析を行うという我々の問題設

定について述べる．

### 2.1 Android における暗号化

Android アプリを開発する場合には，Java 言語または Java と相互運用可能な Kotlin 言語を使用して開発することが一般的であるが，いずれの言語を使用する場合においても，データを暗号化する際には Code 1 に示すように Java の標準的な暗号化 API である Cipher API を使用することが一般的である [12]．

Code 1: Cipher API を用いた暗号化/復号の例

```
public static byte[] encrypt(Key key, byte[]
    data) throws Exception
{
    Cipher cipher = Cipher.getInstance("AES");
    cipher.init(Cipher.ENCRYPT_MODE, key);
    byte[] encrypted = cipher.doFinal(data);
    return encrypted;
}

public static byte[] decrypt(Key key, byte[]
    data) throws Exception
{
    Cipher cipher = Cipher.getInstance("AES");
    cipher.init(Cipher.DECRYPT_MODE, key);
    byte[] decrypted = cipher.doFinal(data);
    return decrypted;
}
```

Code 1 に示した例で分かるように, Cipher API を使用する際には引数として暗号鍵や使用する暗号方式といった複数のパラメータが必要である．Android OS では Cipher API の使用に必要な典型的なパラメータを生成するために複数の API が用意されており，次は Cipher API 及びパラメータ生成に用いられる API について述べる．

#### 2.1.1 Cipher API

Cipher クラスは暗号化及び復号に関して各種の暗号アルゴリズムを使用する API を提供しており，標準状態の Android OS で使用することができる．Cipher クラスを使用する場合，まず始めに getInstance メソッドを呼び出して Cipher クラスのオブジェクトを生成し，その後 init メソッドを呼び出して生成した Cipher オブジェクトの使用方法等を定義することになる．getInstance 及び init メソッドは表 1 に示すような呼び出し方法が存在し，呼び出す際にはいくつかの引数が必要となるが，そのうち暗号鍵や暗号アルゴリズムのパラメータを生成するために使用される標準的な API については次節で述べる．

#### 2.1.2 暗号化パラメータ生成用の API

暗号鍵及び暗号アルゴリズムのパラメータを定義するために，Android OS ではそれぞれ KeySpec クラス及び AlgorithmParameterSpec クラスが用意されており，それぞれのクラスに属する様々な API を使用して Cipher クラスで使用する各種暗号化に必要なパラメータを生成することができる．表 2 にはそれらのパラメータ生成用の API のうち，暗号鍵や初期化ベクトル等の定義に使用されるような代表的なものを示す．また，Code 2 にはパスワードにより生成された暗号鍵を使用してデータを暗号化する場合の例を示す．この場合，Cipher オブジェクトは暗号鍵及び初期化ベクトルとしてそれぞれ PBEKeySpec 及び IvParameterSpec を使用して必要なパラメータを生成している．

Code 2: パスワードベースの暗号化

```
String TRANSFORMATION = "AES/CBC/PKCS7Padding";
String KEY_GEN_MODE =
    "PBEWITHSHA256AND128BITAES-CBC-BC";
int SALT_LENGTH = 20;
int ITER_COUNT = 1024;
int KEY_LENGTH = 128;

char[] passwd = new char[] { ... };
byte[] salt = new byte[] { ... };
byte[] iv = new byte[] { ... };

Cipher cipher =
    Cipher.getInstance(TRANSFORMATION);
PBEKeySpec kSpec = new PBEKeySpec(passwd, salt,
    ITER_COUNT, KEY_LENGTH);
SecretKeyFactory skFactory =
    SecretKeyFactory.getInstance(KEY_GEN_MODE);
SecretKey sKey = skFactory.generateSecret(kSpec);
IvParameterSpec ivSpec = new IvParameterSpec(iv);
cipher.init(Cipher.DECRYPT_MODE, sKey, ivSpec);
```

### 2.2 本稿における問題設定

Code 1 及び 2 で示したとおり，Cipher API を使用して暗号化処理を行うためには，複数の必要なパラメータを生成する必要がある．それらパラメータの生成方法を判明させることができれば，アプリによって暗号化されたデータの解析に必要な復号方法を判明させられる可能性がある．

したがって，本稿での問題は「Cipher API を使用した暗号化処理に関して，必要なパラメータを特定すること」と設定する．Cipher API を使用して暗号化処理を場合には処理のモードを設定する必要があるが，本研究の目的はデジタル・フォレンジックへの活用を目指し，暗号化されたデータの復号方法を得ることであるため，処理のモードとして復号処理を行う Cipher.DECRYPT_MODE で使用されている部分に注目することとする．

表 3 には本稿で取り扱う暗号化 API を示している．これらの API を使用するためには複数のパラメータが必要となるが，代表的なものとしては，暗号鍵等に使用される byte[] 型の値，ユーザーが入力したパスワード等に使用される char[] 型の値，各種必要な文字列や整数値に使用される String や int の値が挙げられる．

すなわち，パラメータ生成の過程を特定するために，次に示す値の導出に注目し，どのような過程によって生成されるかを特定することとする．

- byte[] 型の値

表 1: Cipher クラスの対象の API
**Table 1** Cipher class APIs

| API Method and Return Value | Method Description |
|---|---|
| `Cipher getInstance(String)`<br>`Cipher getInstance(String, String)`<br>`Cipher getInstance(String, Provider)` | Returns a Cipher object |
| `void init(int, Key)`<br>`void init(int, Key, SecureRandom)`<br>`void init(int, Key, AlgorithmParameters)`<br>`void init(int, Key, AlgorithmParameters, SecureRandom)`<br>`void init(int, Key, AlgorithmParameterSpec)`<br>`void init(int, Key, AlgorithmParameterSpec, SecureRandom)` | Initialize the Cipher object |

表 2: パラメータ生成用 API の例
**Table 2** Parameter specification APIs

| Class Description | API Class | Method for Initialization | Parameters Description |
|---|---|---|---|
| Key Specification | `PBEKeySpec` | `void <init>(char[])` | `char[]` : password |
| | | `void <init>(char[], byte[], int)` | `byte[]` : salt |
| | | `void <init>(char[], byte[], int, int)` | `int` : iteration, length |
| | `SecretKeySpec` | `void <init>(byte[], String)` | `byte[]` : key |
| | | `void <init>(byte[], int, int, String)` | `int` : offset, length |
| IV Specification | `IvParameterSpec` | `void <init>(byte[])` | `byte[]` : iv |
| | | `void <init>(byte[], int, int)` | `int` : offset, length |
| Parameters Specification | `PBEParameterSpec` | `void <init>(byte[], int)` | `byte[]` : salt |
| | | `void <init>(byte[], int, AlgorithmParameterSpec)` | `int` : iteration |

- `char[]` 型の値
- `String` 型の値
- `int` 型の値

## 3. 解析手法

### 3.1 概要

前節で述べたように，暗号化 API のパラメータ生成について特定することを目的とし，アプリの解析を行う．具体的には，次に示すような手順である．

(1) `Cipher` の `init` メソッドを `Cipher.DECRYPT_MODE` で呼び出している部分を検索する．
(2) 発見したメソッド呼び出しのそれぞれについて，暗号化 API を使用してパラメータ生成を行っている部分を特定する．
(3) 特定できたそれぞれの暗号化 API について，固定値を使用している部分を特定しその値を得る．
(4) 特定できたそれぞれの暗号化 API について，パラメータのソースとなる API を特定する．

簡単のために，Java のソースコードを用いて解析方法の概要を図 1 に示す．まず Step 1. としてアプリ内で `Cipher` の `init` メソッドが `Cipher.DECRYPT_MODE` で呼び出されている部分を発見すると，Step 2. として当該 `Cipher` オブジェクトの生成過程において必要となった暗号化 API を特定する．図 1 の例では `PBEKeySpec`, `SecretKeyFactory` 及び `IvParameterSpec` が特定される．その後，Step 3. 及び 4. として，それぞれの暗号化 API で使用されたパラメータとして，アプリ内に定義されている固定値やソースとなる API が特定される．

なお，パラメータのソースとなる API については次節においてその詳細を述べる．

### 3.2 パラメータのソース

実際の Android アプリでは，暗号化 API で使用されるパラメータの生成として様々な処理が使用されているが，本稿では生成の元となる値について次の 5 種類の取得方法の分類を定義する．

**Source 1** 固定値として定義された値を取得
**Source 2** デバイスの固有値を取得
**Source 3** ファイルから値を取得
**Source 4** ユーザーによる入力から値を取得
**Source 5** 外部ネットワークからの値の取得

ここからそれぞれの具体的な内容及びそれぞれの分類に該当する API 等について説明する．

**Source 1　固定値として定義された値を取得**

値の取得方法の中でも最も単純な方法として，値をアプリのソースコードにハードコードして定義する方法がある．実例として，Code 3 及び 4 にそれぞれ暗号モードと暗号鍵の生成に使用される値としてハードコードされた

表 3: 対象の暗号化 API
**Table 3** Crypto APIs

| API Class | Method and Parameters | Method Description |
|---|---|---|
| Cipher | Cipher getInstance(String) | Cipher Initialize |
| Cipher | Cipher getInstance(String, String) | |
| Cipher | Cipher getInstance(String, Provider) | |
| Cipher | void init(int, Key, AlgorithmParameters) | |
| Cipher | void init(int, Key, SecureRandom) | |
| Cipher | void init(int, Key, AlgorithmParameterSpec) | |
| Cipher | void init(int, Key) | |
| Cipher | void init(int, Key, AlgorithmParameterSpec, SecureRandom) | |
| Cipher | void init(int, Key, AlgorithmParameters, SecureRandom) | |
| SecretKeyFactory | SecretKeyFactory getInstance(String) | Key Factory |
| SecretKeyFactory | SecretKeyFactory getInstance(String, String) | |
| SecretKeyFactory | SecretKeyFactory getInstance(String, Provider) | |
| DESedeKeySpec | void <init>(byte[]) | Key Specification |
| DESedeKeySpec | void <init>(byte[], int) | |
| DESKeySpec | void <init>(byte[]) | |
| DESKeySpec | void <init>(byte[], int) | |
| PBEKeySpec | void <init>(char[]) | |
| PBEKeySpec | void <init>(char[], byte[], int) | |
| PBEKeySpec | void <init>(char[], byte[], int, int) | |
| SecretKeySpec | void <init>(byte[], String) | |
| SecretKeySpec | void <init>(byte[], IIString) | |
| GCMParameterSpec | void <init>(int, byte[]) | Algorithm Specification |
| GCMParameterSpec | void <init>(int, byte[], int, int) | |
| IvParameterSpec | void <init>(byte[]) | |
| IvParameterSpec | void <init>(byte[], int, int) | |
| PBEParameterSpec | void <init>(byte[], int) | |
| PBEParameterSpec | void <init>(byte[], int, AlgorithmParameterSpec) | |

```
public class DecryptActivity extends Activity {
    String TRANSFORMATION = "AES/CBC/PKCS7Padding";
    String KEY_GEN_MODE = "PBEWITHSHA256AND128BITAES-CBC-BC";
    int SALT_LENGTH = 20;
    int IV_LENGTH = 16;
    int ITER_COUNT = 1024;
    int KEY_LENGTH = 128;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        ...
    }

    public void onAction(View view) {
        TextView textViewPassword = (TextView)findViewById(R.id.editTextPassword);
        char[] password = textViewPassword.getEditableText().toString().toCharArray();

        FileInputStream fileInput = openFileInput(FILENAME);
        byte[] salt = new byte[SALT_LENGTH];
        fileInput.read(salt);
        byte[] iv = new byte[IV_LENGTH];
        fileInput.read(iv);

        Cipher cipher = Cipher.getInstance(TRANSFORMATION);
        PBEKeySpec kSpec = new PBEKeySpec(password, salt, ITER_COUNT, KEY_LENGTH);
        SecretKeyFactory skFactory = SecretKeyFactory.getInstance(KEY_GEN_MODE);
        SecretKey sKey = skFactory.generateSecret(kSpec);
        IvParameterSpec ivSpec = new IvParameterSpec(iv);
        cipher.init(Cipher.DECRYPT_MODE, sKey, ivSpec);
        ...
    }
}
```

Step 2.
Step 1.
Step 3. & 4.

図 1: 解析方法の概要
**Fig. 1** Analysis steps using code example

String 型の値及び byte[] 型の値を使用する例を示す.

```
public static final String TRANSFORMATION =
    "DES/CBC/PKCS5Padding";
```

Code 3: ハードコードされた String 型の値

```
Cipher c = Cipher.getInstance(TRANSFORMATION);
```

Code 4: ハードコードされた byte[] 型の値

```
byte[] sKey = new byte[] {0x00, 0x01, 0x02, 0x03,
    0x04, 0x05, 0x06, 0x07, 0x08, 0x09, 0x0A,
    0x0B, 0x0C, 0x0D, 0x0E, 0x0F};
SecretKeySpec sKeySpec = new SecretKeySpec(sKey,
    "AES");
```

また，Android アプリでは固定値をリソースとしてコードの外部に定義することも可能である．Code 5 では，リソースに整数値を定義して繰り返し回数として使用する例を示す．

Code 5: アプリのリソース内に定義された値及びその取得

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <integer name="iteration">1024</integer>
</resources>
```

```
int iterationCount =
    getResources().getInteger(R.integer.iteration);
```

リソースの定義は xml 形式で保存され，コードからリソースの値を使用するためには固有の API を使用する．そのため，リソースの値を取得するための API である Resources.getInteger 及び Resources.getString をパラメータのソースとして扱うこととする．

**Source 2　デバイスの固有値を取得**

実際のアプリの例として，暗号鍵の生成にデバイスの固有値の 1 つである IMEI を使用しているアプリが存在する [13]．Code 6 には IMEI を取得する API の使用例を示す．

Code 6: IMEI の値の取得

```
TelephonyManager tm = (TelephonyManager)
    getSystemService(Context.TELEPHONY_SERVICE);
String device_id = tm.getDeviceId();
```

本稿ではデバイスの固有値を取得する API として, IMEI, SIM の ID 及び電話番号を取得する API をパラメータのソースとして扱うこととする．

**Source 3　ファイルから値を取得**

アプリで暗号化処理を行う際の典型的な動作として，暗号化する際に必要なパラメータの値をファイルに保存しておき，復号の際にファイルから値を取得するという動作がある．そのため，ファイルを読み込む API をパラメータのソースとして扱う．Android ではファイルを読み込む API が大きく分けて 3 種類あり，Code 7 - 9 に示すようにファイルへの直接のアクセス，SharedPreferences へのアクセス，SQLite ファイルへのアクセスがある．

Code 7: ファイルから byte[] 型の値の読み取り

```
FileInputStream fileInput =
    openFileInput(fileName);
byte[] salt = new byte[16];
fileInput.read(salt);
```

Code 8: SharedPreferences に定義された値の読み取り

```
<?xml version='1.0' encoding='utf-8'
    standalone='yes' ?>
<map>
  <string name="password">Passw0rD</string>
</map>
```

```
SharedPreferences sp =
    getSharedPreferences("DataSave",
    Context.MODE_PRIVATE);
string Password = sp.getString("password", "");
```

Code 9: SQLite ファイルからの読み取り

```
SQLiteDbHelper dbHelper = new
    SQLiteDbHelper(getContext());
SQLiteDatabase db =
    dbHelper.getReadableDatabase();
Cursor c = db.rawQuery("SELECT password FROM
    secret", null);
```

これらの API のうち，それぞれ読み込むための API をパラメータのソースとして扱う．

**Source 4　ユーザーによる入力から値を取得**

暗号化処理を行う際にユーザーによって入力された値を使用する場合があり，典型的なものとしては Code 10 に示すようにパスワードを使用して暗号鍵を生成するといった動作が挙げられる．

Code 10: ユーザーが入力した文字列の取得

```
EditText et =
    (EditText)findViewById(R.id.editTextPassword);
String passwd = et.getText().toString();
PBEKeySpec kSpec = new
    PBEKeySpec(passwd.toCharArray());
```

このようなユーザー入力から値を取得する API をパラメータのソースとして扱う．

**Source 5　外部ネットワークからの値の取得**

スマートフォンの特性として基本的に常時外部ネットワークに接続しているということがあり，Code 11 に示す例のように，暗号化処理の際に必要なデータをネットワーク経由で取得するという可能性がある．

Code 11: ネットワーク経由による外部からのデータ取得

```
URL url = new URL("http://www.android.com/");
HttpURLConnection urlCon = (HttpURLConnection)
    url.openConnection();
InputStream input = new
    BufferedInputStream(urlCon.getInputStream());
```

このような HTTP 経由により値を取得する API をパラメータのソースとして扱う．

表 4: 実験結果 (暗号化 API 使用の特定数)
**Table 4** Experimental Results (Usages of Crypto APIs)

| Class Category | Crypto API | # Usages |
|---|---|---|
| Key Factory | `SecretKeyFactory` | 42 |
| Key Specification | `DESedeKeySpec` | 0 |
| | `DESKeySpec` | 2 |
| | `PBEKeySpec` | 19 |
| | `SecretKeySpec` | 50 |
| Algorithm Specification | `GCMParameterSpec` | 2 |
| | `IvParameterSpec` | 47 |
| | `PBEParameterSpec` | 7 |

### 3.3 パラメータソースと解析の可能性

前節までで示した 5 種類のパラメータソースでは，それぞれで値が取得できる条件が異なってくる．そのうち Source 1 - 3 についてはパラメータがデバイスのみに依存するため，デバイスから復号に必要な全ての情報が得られる．一方，Source 4, 5 の場合には復号にそれぞれユーザーの入力や外部ネットワークからの情報が必要となる．

## 4. 実験と評価

前節までで説明した解析を行うため，Android アプリの静的解析フレームワークである ”Amandroid” [10] を利用して実装を行った．具体的には ”Amandroid” の機能である ”Explicit Value Finder” を拡張してハードコードされた値の特定を行い，”Amandroid” のテイント解析機能を使用し，暗号化 API のパラメータソースとして使用される API の特定を行った．

解析手法の評価を行うため，オープンソースの Android アプリを収集している ”F-Droid”[14] から 2,078 個の実在するアプリ及びそのソースコードを入手し，そのうちソースコード中に `Cipher` API を使用していることを示す`"import javax.crypto.Cipher"` の記述が発見できた 139 個のアプリに対して解析の実験を行った．

### 4.1 結果

実験の結果，復号モードである `Cipher.DECRYPT_MODE` での `Cipher.init` メソッドの呼び出しを合計 200 箇所発見し，それぞれの `Cipher` API についてパラメータとして使用されている暗号化 API を解析し，表 4 に示すように合計で 178 箇所特定した．

その後，特定できた暗号化 API のそれぞれについてパラメータソースを解析し，表 5 に示すように合計で 212 箇所特定した．

## 5. 議論

### 5.1 制限事項

現状の実装では暗号化処理を行う API として `Cipher` API のみに注目して解析を行っているが，本稿の解析手法については他の暗号化処理を行う API に対しても適用できる可能性がある．具体的には Android でよく使用されている ”BouncyCastle”[15] や”SQLCipher” [16] といった API の仕様が分かる暗号化ライブラリについては適用可能と考えられるが，独自の実装等による仕様が不明な場合の対応については今後の課題となる．また，本手法の実装は ”Amandroid” に基づいており，Java Native Interface を利用したネイティブライブラリが暗号化処理に使用されている場合，現在の実装では対応できずこちらも今後の課題である．

### 5.2 関連研究

デジタル・フォレンジックの観点からのスマートフォンアプリの解析に関しては数多くの研究がなされているが [17], [18], [19], [20], [21], [22], [6], [23]，大半が人手による解析を行っている．代表的な手法としては，アプリを実際のデバイス又はエミュレータ等にインストールし，実際にアプリを動作させて挙動を観測して解析するというものがある．そのような手法では解析に時間がかかる上，解析者に十分な解析技術が必要となり，多くのアプリに対応することは難しい側面がある．

そのような背景から，最近ではアプリの自動解析を行う提案が複数なされている [1], [2], [3], [4]．しかしながら，これらの研究は解析によって保存されるデータの種類と場所を特定することを目的としており，データの保存形式の特定は今後の課題とされている．

一方，Android アプリの暗号化処理に関する解析についての研究も複数なされているが [7], [8], [9]，それらの研究の主な目的は暗号化処理の不適切な実装を検出するというアプリ開発者の観点によるものである．例えば，「暗号モードとして ECB モードは使用しない」といったルールをあらかじめ設定し，設定したルールが守られているか自動的に解析を行うといったものである．一方本稿では暗号化されたデータの復号への活用を目指し，暗号化処理における必要なパラメータを全て特定することを目的としているため，これらの既存研究で目的を達成するには不十分である．

## 6. まとめ

デジタル・フォレンジックにとって，データの暗号化は迅速な解析を阻む可能性のある大きな課題の 1 つである．その課題に対応するため，本稿では既存の Android アプリの静的解析フレームワークを拡張し，暗号化 API の利用状況の特定を行う手法について提案した．また，提案手法を使用して実在する Android アプリを解析し，暗号化処理を行う `Cipher` API やその他の暗号化 API の利用状況を特定した．これらの結果は Android アプリによって暗号化されたデータを解析する際に活用することができ，より精密

表 5: 実験結果 (パラメータソースの特定数)
**Table 5** Experimental Results (Source Identification)

| Category / Parameter Types | | | # `byte[]` | # `char[]` | # `String` | # `int` |
|---|---|---|---|---|---|---|
| Locally Obtainable | Source 1 | | 3 | 0 | 60 | 24 |
| | Source 2 | | 0 | 0 | 0 | 0 |
| | Source 3 | Files | 17 | 1 | 2 | 1 |
| | | SharedPreferences | 53 | 4 | 0 | 0 |
| | | SQLite | 16 | 4 | 0 | 0 |
| User defined value | Source 4 | | 9 | 14 | 0 | 0 |
| Obtainable via network | Source 5 | | 4 | 0 | 0 | 0 |

な解析を行うことが可能となる.

## 参考文献


[1] Lin, X., Chen, T., Zhu, T., Yang, K. and Wei, F.: Automated forensic analysis of mobile applications on Android devices, *Digital Investigation*, Vol. 26, pp. S59 – S66 (online), DOI: https://doi.org/10.1016/j.diin.2018.04.012 (2018).

[2] Cheng, C. C.-C., Shi, C., Gong, N. Z. and Guan, Y.: EviHunter: Identifying Digital Evidence in the Permanent Storage of Android Devices via Static Analysis, *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, CCS '18, New York, NY, USA, ACM, pp. 1338–1350 (online), DOI: 10.1145/3243734.3243808 (2018).

[3] Xu, Z., Shi, C., Cheng, C. C. C., Gong, N. Z. and Guan, Y.: A dynamic taint analysis tool for android app forensics, *Proceedings - 2018 IEEE Symposium on Security and Privacy Workshops, SPW 2018*, pp. 160–169 (2018).

[4] Anglano, C., Canonico, M. and Guazzone, M.: The Android Forensics Automator (AnForA): A tool for the Automated Forensic Analysis of Android Applications, *Computers and Security*, Vol. 88, p. 101650 (2020).

[5] Paul, G. and Irvine, J.: Investigating the security of android security applications, *9th CMI Conference on Smart Living, Cyber Security and Privacy* (2016).

[6] Zhang, X., Baggili, I. and Breitinger, F.: Breaking into the vault: Privacy, security and forensic analysis of Android vault applications, *Computers and Security*, Vol. 70, pp. 516–531 (2017).

[7] 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, New York, USA, ACM Press, pp. 73–84 (online), DOI: 10.1145/2508859.2516693 (2013).

[8] Shuai, S., Guowei, D., Tao, G., Tianchang, Y. and Chenjie, S.: 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 (online), DOI: 10.1109/DASC.2014.22 (2014).

[9] Kruger, S., Spath, J., Ali, K., Bodden, E. and Mezini, M.: CrySL: An Extensible Approach to Validating the Correct Usage of Cryptographic APIs, *IEEE Transactions on Software Engineering*, Vol. PP, pp. 1–1 (2019).

[10] Wei, F., Roy, S., Ou, X. and Robby: Amandroid: A Precise and General Inter-component Data Flow Analysis Framework for Security Vetting of Android Apps, *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, CCS '14, New York, NY, USA, ACM, pp. 1329–1341 (online), DOI: 10.1145/2660267.2660357 (2014).

[11] 一般社団法人日本スマートフォンセキュリティ協会（JSSEC）セキュアコーディング WG：Android アプリのセキュア設計・セキュアコーディングガイド 2019 年 12 月 1 日版, `https://www.jssec.org/dl/android_securecoding/` (2019).

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

[13] Focus, F.: How To Decrypt WeChat EnMicroMsg.db Database? — Forensic Focus - Articles, `https://articles.forensicfocus.com/2014/10/01/decrypt-wechat-enmicromsgdb-database/` (2014).

[14] F-Droid: F-Droid - Free and Open Source Android App Repository, `https://f-droid.org` (2020).

[15] of the Bouncy Castle Inc., L.: bouncycastle.org, `https://www.zetetic.net/sqlcipher/` (2020).

[16] Zetetic, L.: SQLCipher - Zetetic, `https://www.bouncycastle.org/` (2020).

[17] I.Al-Saleh, M. and A. Forihat, Y.: Skype Forensics in Android Devices, *International Journal of Computer Applications*, Vol. 78, pp. 38–44 (2013).

[18] Mehrotra, T. and Mehtre, B. M.: Forensic analysis of Wickr application on android devices, *2013 IEEE International Conference on Computational Intelligence and Computing Research, IEEE ICCIC 2013*, pp. 1–6 (2013).

[19] Anglano, C.: Forensic analysis of whats app messenger on Android smartphones, *Digital Investigation*, Vol. 11, No. 3, pp. 201–213 (online), DOI: 10.1016/j.diin.2014.04.003 (2014).

[20] Anglano, C., Canonico, M. and Guazzone, M.: Forensic analysis of the ChatSecure instant messaging application on android smartphones, *Digital Investigation*, Vol. 19, pp. 44–59 (online), DOI: 10.1016/j.diin.2016.10.001 (2016).

[21] Wu, S., Zhang, Y., Wang, X., Xiong, X. and Du, L.: Forensic analysis of WeChat on Android smartphones, *Digital Investigation*, Vol. 21, pp. 3–10 (online), DOI: 10.1016/j.diin.2016.11.002 (2017).

[22] Anglano, C., Canonico, M. and Guazzone, M.: Forensic analysis of Telegram Messenger on Android smartphones, *Digital Investigation*, Vol. 23, pp. 31–49 (2017).

[23] Alyahya, T. and Kausar, F.: Snapchat Analysis to Discover Digital Forensic Artifacts on Android Smartphone, *Procedia Computer Science*, Vol. 109, pp. 1035–1040 (2017).