Vol.2020-CSEC-90 No.25
Vol.2020-SPT-38 No.25
2020/7/21

Androidアプリケーションにおける暗号API利用動向の基礎調査

河合 惇丞[1]　金岡 晃[1,a)]

概要：開発者向けユーザブルセキュリティ，ユーザブルプライバシーの研究分野においてソフトウェア開発者の暗号技術の利用に関する研究が活発になっている．それらの研究でそういった暗号技術の利用が適切にされておらず脆弱性が存在するソフトウェアが多数あることがわかってきた．しかしこれらの研究はSSL/TLSやDES，AESなど特定の暗号技術に限った調査だけであり，暗号技術全般の網羅的な調査が行われていない．Androidアプリケーションは SSL/TLS をはじめとして暗号技術の利用がされているケースが多くある．そこで本研究ではソフトウェアの暗号技術の利用状況の網羅的な分析の基礎調査として，Androidアプリケーションを調査対象としその暗号 API の利用状況を調査した。Androidアプリケーションを静的解析し，暗号で用いられるメソッド名や特徴のある用語によるフィルタリング，アルゴリズムが指定可能な代表的箇所の抽出，API の利用傾向分析により，現状どの程度暗号技術が利用されているのか，その時のアルゴリズムはどのようなものが多く利用されているのかなどを探った．

1. はじめに

近年開発者向けユーザブルセキュリティ，ユーザブルプライバシーの研究分野においてソフトウェア開発者の暗号技術の利用に関する研究が活発になっている．中でも暗号API の誤使用に関する研究が最も多く，Egele らの研究 [1] や Lazar らの研究 [2] がある．暗号 API の誤使用への対策として，ソフトウェア開発用の統合開発環境のプラグインの提案がされている [3], [4]．また，暗号 API の誤使用の原因把握としての研究も行われており，StackOverflow のような開発者向けフォーラムの影響により開発者がソースコードの一部を再利用することで脆弱性が広まる可能性があることが指摘されている [5], [6], [7].

それらの研究で暗号技術の利用が適切にされておらず脆弱性が存在するソフトウェアが多数あると分かっている．しかしこれらの研究はSSL/TLSやDES，AESなど特定の暗号技術，ソフトウェア開発者の開発時の誤使用など特定の状況に限った調査だけであり，実際にどの程度のソフトウェアでどのように暗号技術が利用されているのかなどの暗号技術全般の網羅的な調査は行われていない．

Androidアプリケーションにおいても同様に SSL/TLS をはじめとして暗号技術の利用がされているケースが多くある．そこで本研究ではソフトウェアの暗号技術の利用状況の網羅的な調査の一環として，Apktool や baksmali といったツールによる静的解析が容易であること，世界のモバイル端末における OS のシェア率が高いこと?からJava で開発された Android アプリケーションを調査対象とする．

Androidアプリケーションを静的解析し，暗号で用いられるメソッド名や特徴のある用語によるフィルタリングや，アルゴリズムが指定可能な代表的箇所の抽出，API の利用傾向分析を行う．Androidアプリケーションの現状はどのような暗号技術が利用され，どの程度暗号技術が利用されているのか，その時のアルゴリズムはどのようなものがどれ程利用されているのか，また暗号技術の利用には何らかの傾向があるのかなどを明らかにすることを目的とする．

調査には Android アプリケーションのサードパーティ配布ストアの 1 つである AndroZoo より取得した 411,486 個の Android アプリケーションと開発者向け公式サイト Android Developers の API リファレンスに記載されている API から抽出した暗号・セキュリティに関するメソッド 4,324 個のリストを使用する．

調査の結果，調査対象の Android アプリケーション群における暗号・セキュリティに関するメソッドの利用数，利用率，またアルゴリズムが指定可能な代表的な箇所において指定されているアルゴリズム，メソッド同士の関係性が判明した．

[1] 東邦大学
Toho University, Funabashi, Chiba 274–8510, Japan
[a)] akira.kanaoka@is.sci.toho-u.ac.jp

2. 関連研究

2.1 Egele らの調査

Egele らは Android アプリケーションにおける暗号 API の誤使用があることに注目し，その誤使用の解析を行う軽量手法を提案した [1]．そこで解析対象とされている項目は以下のものとなっていた．

- 暗号化に ECB モードを使っていないか
- CBC モードにおいて非ランダムな IV を使っていないか
- 暗号鍵が固定でないか
- パスワードベースの暗号化において Salt が固定出ないか
- パスワードベースの暗号化において 1000 回未満の繰り返しになっていないか
- SecureRandom 関数利用時に静的な Seed を使っていないか

2.2 Lazar らの調査

Lazar らによる調査 [2] は，2011 年から 2014 年にかけて明らかになった暗号関連の脆弱性 269 件を調査し，その内訳を示した．その結果，17%がライブラリ自身のバグであり，83%が暗号の誤使用であることが明らかにされた．

2.3 Li らの調査

Li らによる調査 [8] は，Apple 社が iPhone などに向けて提供する iOS のアプリケーションを調査対象とし，静的解析と動的解析を組み合わせて暗号 API の利用を追跡する iCryptoTracer を提案した．調査対象の 98 個のアプリケーションのうち，64 個のアプリケーションで暗号の誤使用が確認されたとしている．

2.4 角田らの調査

角田らの調査では，デジタル・フォレンジックと呼ばれるデジタルデバイスに対する情報の収集や分析調査についての視点から暗号技術の利用状況を明らかにすることの必要性を指摘し，Android アプリケーションにおけるその静的解析手法の提案をした．[9] しかし，調査対象の暗号技術が javax.crypto.Cipher クラスの getInstance メソッドと init メソッドであり，暗号技術の網羅的なものではなかった．

3. 調査の方針と方法

4. 分析の手法

本研究では，静的解析が容易であること，世界でのモバイル端末における OS のシェア率が高いことから Java で開発された Android APK を調査対象とする．

4.1 APK で利用される暗号技術について

APK で利用される暗号技術は大きく分けて以下の 3 つに分類することができる．

- Android の開発者向け公式 Web サイトである Android Developers の API リファレンス [10] に記載されている API
- サードパーティ製の API
- 独自実装等の API

これらについて以下で説明を行う．

4.1.1 リファレンス API

Android 公式の開発者向け Web サイトである Android Developers では APK で利用することができる API パッケージが記載されてる．Java で利用することができる基本的な API パッケージの他に Android 独自の android から始まるパッケージ名を持つ API パッケージなどがある．

4.1.2 サードパーティ製の API について

サードパーティ製の API として Google 社の tink[11] や Facebook 社の Conceal[12] などが知られている．企業が提供しているものや開発者が提供しているものなど，サードパーティ製の暗号・セキュリティ API は多岐にわたる．GitHub で公開されているリポジトリで “android”，“cryptography” タグを含むものは 2020 年 1 月 14 日時点で 180 に上る．

4.1.3 独自実装等の API について

本研究では，APK 開発者が既存の API を利用せず独自に実装した API や上記の 2 つに含まれないものを独自実装等の API とする．2015 年の Reaves らの論文でもその存在が指摘されている [13]．

4.2 APK における暗号技術利用動向調査手法

この章では，APK における暗号技術利用動向調査のための分析手法について説明する．

4.2.1 利用されている API の分析

Android Developers の API リファレンスに記載されている API から，暗号・セキュリティに関するパッケージ，クラス，メソッドを抽出しリスト化する．このリストをもとに APK においてどれほど暗号技術が利用されているかの分析を行う．調査対象となる APK 群は AndroZoo[14] より取得する．APK 群を静的解析し、APK で呼び出されているメソッド名や引数の情報と Android Developers の API リファレンスから作成したリストをマッチングさせることで，APK で利用されている API を分析する．

4.2.2 アルゴリズムが指定可能であるメソッドにおける，指定されたアルゴリズムの分析

java.security.MessageDigest.getInstance はメッセージダイジェストアルゴリズムの機能を提供する MessageDigest クラスのメソッドで，引数にアルゴリズムを指定することで指定されたメッセージダイジェストアルゴ

表 1 java.security.MessageDigest.getInstance(java/lang/String) の引数で指定可能なアルゴリズム一覧

アルゴリズム名
MD5
SHA-1
SHA-224
SHA-256
SHA-384
SHA-512

リズムを実装した MessageDigest オブジェクトを返す．java.security.MessageDigest.getInstance の引数で指定可能なアルゴリズムを表 1 に示す．

API の中には java.security.MessageDigest.getInstance のように引数にアルゴリズムを指定可能なメソッドがある．APK 開発者によって，引数にアルゴリズムを指定可能なメソッドを利用した際に脆弱性を含むアルゴリズムが指定されてしまう可能性がある．この分析により，現状どのようなアルゴリズムが利用されているのかを明らかにする．

まず，アルゴリズムが指定可能な代表的箇所を見つけ，その代表的箇所を含む APK を調査対象とし、設定されたアルゴリズムを抽出する．

アルゴリズムを指定する方法は，メソッド宣言時に引数内に直接アルゴリズムを指定する等のメソッドの宣言をしたクラスと同一クラス内でアルゴリズムを宣言する方法とメソッドの宣言の時に別のクラスで宣言したアルゴリズムを指定する方法の二つに大別される．前者の方法は同一 smali ファイルにメソッドと引数の情報が記述されるのに対し，後者の方法ではメソッドの情報と引数の情報が別々の smali ファイルに記述される．本研究では解析が容易である前者の方法で指定されたアルゴリズムの調査を行う．

4.2.3 利用される API の傾向の分析

メソッド A を利用した場合，およそすべての場合においてメソッド B を利用するという状況があるとき，メソッド A を利用したにも関わらずメソッド B を利用していなかった場合，APK 開発者のミスによりメソッド B を利用しなかった場合と APK 開発者がメソッド B に代わるメソッド C を利用した場合が考えられる．この場合メソッド C は APK 開発者がサードパーティ製の API を利用した可能性と独自にメソッド C を実装した可能性がある．その前段階の分析として，メソッド A が利用された時のメソッド B の利用率を明らかにする．

調査対象の APK 群の各 APK 内で利用されている API からメソッド A とメソッド B という 2 つのメソッドの組み合わせを総当たりで作成し，調査対象の APK 全ての組み合わせをリスト化し集計する．4.2.1 の結果と集計した結果を用いてメソッド A が利用された時のメソッド B の利用率 $P(B|A)$ を明らかにする．

表 2 Android Developers の API リファレンスに記載される API より取得した暗号・セキュリティに関するクラスが持つ 4,324 個のメソッドのリストの一部抜粋

クラス名	メソッド名 (引数)
android.security.KeyChain	KeyChain()
java.security.KeyStore	getProvider()
javax.crypto.Cipher	getInstance(java.lang.String)

4.3 サードパーティ製の API の分析

Android Developers に記載されている API とは違いサードパーティ製の API には API のドキュメントが作成されていないものがある．サードパーティ製の API の分析ではまず，その API についてのドキュメントが存在しているかの確認を行い，存在する場合はそのドキュメントから API のリストを作成する．存在しない場合はサードパーティ製の API のソースコードを解析し，API のドキュメントを作成してから API のリストの作成を行う．APK の解析の前に API の解析を行う必要があるという点が Android Developers に記載されている API との違いである．作成したリストをもとに 4.2 と同様の分析を行うことが可能であると考える．本分析は今後の課題とする．

4.4 独自実装等の API の分析

独自実装等の API は，ドキュメントが作成され，かつ公開されている可能性が低いため，4.2 や 4.3 の様に API のドキュメントから API のリストを作成することができない．RSA や ECC，Crypto といった暗号・セキュリティに関するキーワードを API のリストの代わりとし検索することで独自実装等の API を発見できると考える．4.2.3 の結果より，メソッド A が利用された時のメソッド B の利用率 $P(B|A)$ がわかる．$P(B|A)$ が高いなかでメソッド B を利用していない APK はメソッド B を独自実装等の API で代用している可能性がある．このような APK が主な分析対象となる．本分析は今後の課題とする．

5. 分析を行う実験の手法

5.1 利用されている API の分析

5.1.1 調査を行う API の取得

調査を行う API として Android Developers の API リファレンスに記載される API より暗号・セキュリティに関するクラスを著者らが選別し，そのクラスが持つメソッド計 4,324 個のリスト化を行った．リスト作成は 2019 年 6 月に行い、その時点での API レベルは 29 であった．この時，暗号・セキュリティに関するメソッドに加え SSL や TLS との関係性を考慮しネットワークに関係のあるメソッドもリストに加えた．リストの一部を抜粋し，表 2 に示す．

5.1.2 APK の取得

APK において利用される API の調査対象として，AndroZoo のデータセットより APK を 411,486 個用意した．

表 3　アルゴリズムが指定可能な getInstance メソッドの例

メソッド名 (引数)
java.security.MessageDigest.getInstance(java.lang.String)
javax.crypto.Cipher.getInstance(java.lang.String)
javax.net.ssl.SSLContext.getInstance(java.lang.String)

5.1.3 APK 内のクラス・メソッド情報とと調査対象 API のマッチング

5.1.1 で説明した API リスト (以後 API リスト) と取得した APK ファイルで利用されるメソッドのマッチングは，APK ファイル静的解析の結果得られたメソッド呼び出し情報と API 名のマッチングにより実現する．マッチングはクラス名，メソッド名，引数の数で行う．API リストの中にはクラス名，メソッド名，引数の数が全く同じだが引数の型が異なるものがある．その場合はマッチングの仕方をクラス名，メソッド名，引数がすべて一致するかどうかに変える．

5.2 アルゴリズムが指定可能なメソッドにおける，利用アルゴリズムの分析

5.2.1 アルゴリズム名が指定可能な代表的箇所

アルゴリズムが指定可能な代表的箇所として getInstance メソッドを選択した．暗号・セキュリティに関するクラスの各 getInstance メソッドはアルゴリズムを指定可能な場合が多い．表 3 にその一部を示す．また，4.2.1 の結果より調査対象 APK 群の中で 1 番利用率の高いアルゴリズムを指定可能なメソッドは java.security.MessageDigest.getInstance(java.lang.String) であった．

5.2.2 調査対象 APK の getInstance メソッドの引数の取得

Java においてメソッドの引数の値はそのメソッドの宣言と同時かその宣言以前に宣言される．メソッド宣言と同時に引数が記載されている場合は引数情報の取得は容易であるが、たとえば別のクラスや設定ファイルで宣言されるなどしている場合は、追跡が必要になる．本研究では宣言と同時に引数が記載されている場合だけを調査した．

5.3 利用される API の傾向の分析

調査対象の APK 群においてあるメソッド A が 1 回でも利用されている確率を $P(A)$ とすると，あるメソッド A が 1 回でも利用された時のあるメソッド B が 1 回でも利用される確率は，$P(B|A) = \frac{P(A \cap B)}{P(A)}$ となる．$P(A)$ は 5.1.3 でわかってるため，本分析では $P(A \cap B)$ を調査した．

6. API 情報の整理

表 4　静的解析を行った APK 群の情報

APK 総数	411,486
静的解析に成功した APK 数	401,971
暗号・セキュリティ関連 API 利用 APK 数	280,505

7. 調査の結果と考察

7.1 利用されている API の分析

調査対象の APK 群の smali ファイル化の成否と API 利用の概要を表 4 に示す．smali ファイル化に失敗した要因は APK の難読化や AndroidManifest.xml より取得した起動パッケージに含まれるクラスの欠損等が考えられる．

暗号・セキュリティに関する各 API における調査対象の APK 群の中でその API を少なくとも 1 回は利用した回数とその API が少なくとも 1 回は利用されている確率の上位 10 件を表 5 に示す．暗号・セキュリティに関するメソッドとして，java.security.MessageDigest.digest() の 43,418 個が最も利用されており，次点は java.security.MessageDigest.getInstance(java.lang.String) の 37,405 個であった．java.security.MessageDigest クラスは主に SHA-1 や SHA-256 といったアルゴリズムを使用したハッシュ値を提供するものである．調査対象の APK 群のうち 10.80%がハッシュ値を利用していることがわかる．また，javax.crypto.spec.SecretKeySpec.SecretKeySpec(byte[],java.lang.Strin が表 5 でメッセージダイジェストに関するクラスの次に利用数が多い．javax.crypto.spec.SecretKeySpec クラスは秘密鍵に関する機能を提供するクラスである．調査対象の APK 群において最も利用されている暗号化方式は公開鍵暗号であることが考えられる．

7.2 アルゴリズムが指定可能なメソッドにおける，指定されたアルゴリズムの分析

getInstance メソッド呼び出し時の引数とその使用回数の上位 10 件を表 6 に示す．表 6 中の“取得不可”とある項目は getInstance メソッド呼び出し時の引数が別クラス等で宣言され取得できなかったため取得不可としている．

表 6 より，getInstance メソッドの呼び出し時において，メソッドの引数に直接記述または，メソッドと同一クラスで宣言され指定されたアルゴリズムの中で 1 番利用されていたものは MD5 であることがわかった．表 5 の java.security.MessageDigest クラスの利用数が多いことから，ハッシュ化に関わるアルゴリズムの利用数は多いと考えることができ，これは本実験の結果における MD5 や SHA 等の利用数の多さの理由の一つと考えられる．現在 MD5 の利用は推奨されておらず，SHA-2 以上の強度を持つアルゴリズムの使用が推奨されている．しかし，MD5 の利用数が SHA-256 や SHA-1 より多いことは調査対象の APK 群において利用されるアルゴリズムの更新が遅れて

表 5 暗号・セキュリティに関する各 API における調査対象の APK 群の中でその API を少なくとも 1 回は利用した回数とその API が少なくとも 1 回は利用されている確率の上位 10 件

メソッド名	APK 数 (個)	メソッド利用確率 P(A)(%)
java.security.MessageDigest.digest()	43,418	10.80
java.security.MessageDigest.getInstance(java.lang.String)	37,405	9.31
java.security.MessageDigest.reset()	28,357	7.05
java.security.MessageDigest.update(byte[],int,int)	20,004	4.98
javax.crypto.spec.SecretKeySpec.SecretKeySpec(byte[],java.lang.String)	15,829	3.94
java.security.KeyFactory.generatePublic(java.security.spec.KeySpec)	15,009	3.73
java.security.spec.X509EncodedKeySpec.X509EncodedKeySpec(byte[])	14,662	3.65
java.security.SecureRandom.SecureRandom()	13,759	3.42
java.security.MessageDigest.digest(byte[])	13,414	3.34
javax.crypto.Cipher.doFinal(byte[])	13,009	3.24

いる現状を表していると考えられる．また，利用数自体は MD5 に比べ少ないものの利用されている SHA-1 も MD5 同様に利用は推奨されない．利用が推奨されないアルゴリズムが利用される可能性として，開発者が推奨されていないことを知らずに実装してしまうことと，APK のアップデートがされておらず，推奨されていた当時のままになっていることの 2 つが考えられる．いずれにせよ, 暗号学的ハッシュ関数の利用であれば速やかに利用しているアルゴリズムの変更の必要がある．

本研究のアルゴリズム指定可能なメソッドにおける指定されたアルゴリズムの取得手法は, 引数が宣言されたメソッドと同一クラス内での宣言またはメソッドの宣言時に直接宣言でなければ取得することができない．つまり表 6 における“取得不可”はメソッドを宣言したクラスとは別のクラスで宣言された可能性や引数が動的に変更されるように設定ファイル等が記述されている可能性などが考えられる．また，取得したアルゴリズムはプログラムのソースコードに直接記述されたアルゴリズムであるため，推奨されるアルゴリズムへの変更は速やかに行わなければならないものの，設定ファイル等の軽微な変更ではないため容易ではないという問題点があることがわかった．

getInstance メソッド呼び出し時の引数の中で公開鍵暗号で利用されるアルゴリズムとその使用回数の上位 10 件を表 7 に示す．getInstance メソッド呼び出し時の引数の中で共通鍵暗号で利用されるアルゴリズムとその使用回数の上位 10 件を表 8 に示す．公開鍵暗号では RSA が多く利用されており，共通鍵暗号では AES が多く利用されている．RSA や AES などのアルゴリズムを指定していることはわかっているが，その中でも安全性に関わる情報である鍵長については本実験では調査できていないところであるため，安全であるとは断定できない．

表 6 getInstance メソッド呼び出し時の引数とその使用回数の上位 10 件

引数	利用数 (回)	利用率 (%)
取得不可	64,405	16.02
MD5	50,928	12.67
TLS	16,216	4.03
RSA	14,400	3.58
SHA1withRSA	9,867	2.45
AES/CBC/PKCS5Padding	7,257	1.81
AES	6,175	1.54
SHA-256	4,886	1.22
SHA	4,404	1.10
SHA1	3,524	0.88

表 7 getInstance メソッド呼び出し時の引数の中で公開鍵暗号で利用されるアルゴリズムとその使用回数の上位 10 件

引数 1	引数 2	利用数 (回)
RSA		14,400
SHA1withRSA		9,867
SHA1WithRSA		1,577
RSA/ECB/PKCS1Padding		950
RSA	AndroidKeyStore	595
RSA/ECB/OAEPWithSHA-256AndMGF1Padding		522
RSA	BC	177
RSA/ECB/PKCS1PADDING		63
DSA		52
RSA/NONE/PKCS1Padding		50

7.3 利用される API の傾向の分析

調査対象のメソッドの組み合わせは 876,146 組であった．暗号・セキュリティに関する API 同士の各メソッドの組み合わせにおいてその組み合わせを少なくとも 1 回は利用した APK の数の上位 10 件とその $P(B|A)(\%)$ を表 9 に示

表 8　getInstance メソッド呼び出し時の引数の中で共通鍵暗号で利用されるアルゴリズムとその使用回数の上位 10 件

引数 1	引数 2	利用数 (回)
AES/CBC/PKCS5Padding		7,257
AES		6,175
DES		3,054
AES/ECB/PKCS5Padding		1,665
PBEWITH SHAAND256BIT AES-CBC-BC		1,625
AES/CBC/PKCS7Padding		1,373
DES/CBC/PKCS5Padding		725
AES/CBC/NoPadding		716
AES	AndroidKeyStore	462
DESede		278

す．調査対象の APK 群で利用されるメソッドの組み合わせにおいて全体の傾向として $P(B|A)$ は高いが，メソッド A は利用しているがメソッド B を利用していない APK では，そのメソッド B に代わる機能を提供する API を利用している可能性が考えられる．

8. 今後の課題

8.1 アルゴリズム名が指定可能な箇所の更なる調査

Android アプリケーションで利用される暗号・セキュリティに関する API は Android Developers に記載されている公式の API，サードパーティ製の API，独自実装等の API の 3 つに分類することができる．本研究では Android Developers に記載されている公式の API についての調査以外は行っていない．暗号利用動向の網羅的調査のために今後これらの更なる調査，分析が必要である．

4.2.2 において getInstance メソッドを本研究におけるアルゴリズムを指定可能な代表的な箇所としたが，getInstance メソッド以外にも java.security.MessageDigest クラスや javax.crypto.spec.SecretKeySpec クラスのコンストラクタなど調査を行っていない指定可能な箇所はある．getInstance メソッド以外のアルゴリズムが指定可能な箇所を調査し特定することで 4.2.2 と同様の手法で分析が可能であると考える．

アルゴリズムを指定する方法は，メソッド宣言時に引数内に直接アルゴリズムを指定する等のメソッドの宣言をしたクラスと同一クラス内でアルゴリズムを宣言する方法とメソッドの宣言の時に別のクラスで宣言したアルゴリズムを指定する方法の二つに大別される．本研究では，後者の調査を行っていない．記述されるメソッドを宣言した際の情報の中に含まれるレジスタの情報を遡り構文解析を行うことで別クラス等で指定されたアルゴリズムを取得可能であると考える．

8.2 サードパーティ製の API の調査

4.3 で述べた通り，Android Developers に記載されている API とは違いサードパーティ製の API には API のドキュメントが作成されていないものがある．サードパーティ製の API の分析ではまず，その API についてのドキュメントが存在しているかの確認を行い，存在する場合はそのドキュメントから API のリストを作成する．存在しない場合はサードパーティ製の API のソースコードを解析し，API のドキュメントを作成してから API のリストの作成を行う．APK の解析の前に API の解析を行う必要があるという点が Android Developers に記載されている API との違いである．作成したリストをもとに 4.2 と同様の分析を行うことが可能であると考える．

8.3 独自実装等の API の調査

4.4 で述べた通り，独自実装等の API は，ドキュメントが作成され，かつ公開されている可能性が低いため，4.2 や 4.3 の様に API のドキュメントから API のリストを作成することができない．API のリストではなく開発者が独自実装等の際に利用する可能性の高いキーワード等を調査，整理し調査する．そしてその調査対象のキーワード群がどのメソッド名や引数として利用されているのかを調査しさらにそれらを利用しているメソッドやクラスを発見することで独自実装等の API を調査することができると考える．

8.4 特定 APK 群の特徴調査

7.2 より MD5 等のアルゴリズムを利用する脆弱性を含む APK の存在が確認された．これらの APK の更なる調査として以下の特徴による調査が考えられる．

- APK 配布マーケット
  - AndroZoo より取得した APK には 1 つ 1 つにその APK は AndroZoo がどの APK 配布マーケットから取得したのかという情報があり，この情報により APK をさらに分類することができると考えられる
- APK 公開日
  - その APK がいつ公開されたのかという情報により，その APK が脆弱性を含む理由等の特定につながる可能性がある
  - 公開日の情報は AndroZoo に存在する
- 事業者，利用ライブラリ
  - その APK に関わる事業者や利用しているライブラリの調査によって新たな特徴を発見し更なる暗号利用状況の解明につながる可能性がある
  - APK 静的解析によっては import されたパッケージが別の情報として存在することがある．これにより利用ライブラリ等を調査することが可能である

表 9　暗号・セキュリティに関する API 同士の各メソッドの組み合わせにおいてその組み合わせを少なくとも 1 回は利用した APK の数の上位 10 件とその $P(B|A)(\%)$

メソッド A	メソッド B	APK 数 (個)	$P(B\|A)(\%)$
java.security.MessageDigest.getInstance (java.lang.String)	java.security.MessageDigest.digest()	30,828	82.42
java.security.MessageDigest.digest()	java.security.MessageDigest.getInstance (java.lang.String)	30,828	71.00
java.security.MessageDigest.reset()	java.security.MessageDigest.digest()	26,872	94.76
java.security.MessageDigest.digest()	java.security.MessageDigest.reset()	26,872	61.89
java.security.MessageDigest.getInstance (java.lang.String)	java.security.MessageDigest.reset()	22,495	60.14
java.security.MessageDigest.reset()	java.security.MessageDigest.getInstance (java.lang.String)	22,495	79.33
java.security.MessageDigest.update(byte[],int,int)	java.security.MessageDigest.digest()	19,985	99.91
java.security.MessageDigest.digest()	java.security.MessageDigest.update(byte[],int,int)	19,985	46.03
java.security.spec.X509EncodedKeySpec .X509EncodedKeySpec(byte[])	java.security.KeyFactory.generatePublic (java.security.spec.KeySpec)	14,661	99.99
java.security.KeyFactory .generatePublic(java.security.spec.KeySpec)	java.security.spec.X509EncodedKeySpec .X509EncodedKeySpec(byte[])	14,661	97.68

8.5　同一 APK における変更履歴の調査

AndroZoo は長期間にわたって APK の収集を行っており，同一のアプリケーションであるがバージョンが違うアプリケーションが含まれている可能性がある．それらのバージョンごとの更新による違いなどをを調査し変更履歴を解明することで真にそのアプリケーションが脆弱性を含むかどうかを判断することが可能であると考える．

8.6　利用ライブラリのチェックリストと，ライブラリ利用 APK の暗号利用状況の比較

APK の開発に利用されるライブラリは多岐にわたる．ライブラリによってそのライブラリのドキュメントやサンプルのコードが充実しているものとしていないものがある．ライブラリをチェックリスト化することで，ライブラリのドキュメントが充実しているか否かでそのライブラリを利用している APK にどのような特徴が存在するのかを調査する．この調査により，ライブラリのドキュメントの不備に起因する APK の脆弱性を発見することができると考える．

8.7　暗号技術が正しく利用されているかどうかの調査

Lazar らの調査や Li らの調査によってソフトウェア開発者の暗号技術の誤使用が確認されており，それによってアプリケーションが脆弱性を含むことがわかっている．調査対象の APK 群において利用しているアルゴリズム，暗号技術を利用する際のパラメータ，APK 内の暗号鍵が固定ではないか，ランダムな数字を生成する際のシードはどのようなものか等を調査することで暗号技術が正しく利用されているのかの解明につながると考える．

9.　まとめ

本研究は開発者向けユーザブルセキュリティ，ユーザブルプライバシーの研究分野の現状から暗号技術の網羅的な調査の必要性を考え，その一環として Android アプリケーションの暗号技術利用動向の網羅的な調査を目的とした．APK を AndroZoo より取得し静的解析することで暗号技術の利用動向を調査した．公式の API リファレンスより暗号・セキュリティ・ネットワークに関するメソッドを抽出し、4,324 個の項目を持つリストを作成した．取得した APK は 411,486 個であり，専用ツールを用いて 401,971 個の APK を静的解析した．静的解析結果と作成したリストをマッチングさせることで暗号技術利用の現状を明らかにした．その結果, 69.78%の APK で暗号・セキュリティ・ネットワークに関する API を利用していることがわかった．また，アルゴリズムを指定できる代表的な箇所として getInstance メソッドに焦点を当て，メソッド宣言時に指定されたアルゴリズムの抽出を行った．指定されたアルゴリズムのうち取得できたもので最も利用されていたものは，利用数 50,928 回の現在利用が推奨されていない MD5 であった．さらに利用される API の傾向の分析を行い，今後，より網羅的な調査のために APK 内部の構文解析やサードパーティ製の API，独自実装等の API の調査等の詳細な調査が必要であると考察した．

参考文献

[1] Manuel Egele, David Brumley, Yanick Fratantonio, and Christopher Kruegel. 2013. An empirical study of cryptographic misuse in android applications. In Proceedings of the 2013 ACM SIGSAC confer-

ence on Computer & communications security(CCS ’13). ACM, New York, NY, USA, 73-84. DOI: http://dx.doi.org/10.1145/2508859.2516693

[2] David Lazar, Haogang Chen, Xi Wang, and Nickolai Zeldovich. 2014. Why does cryptographic software fail?: a case study and open problems. In Proceedings of 5th Asia-Pacific Workshop on Systems (APSys’14). ACM, New York, NY, USA, , Article 7 , 7 pages. DOI=http://dx.doi.org/10.1145/2637166.2637237

[3] Steven Arzt, Sarah Nadi, Karim Ali, Eric Bodden, Sebastian Erdweg, and Mira Mezini. 2015. Towards secure integration of cryptographic software. In 2015 ACM International Symposium on New Ideas, New Paradigms, and Reflections on Programming and Software (Onward!) (Onward! 2015). ACM, New York, NY, USA, 1-13. DOI: http://dx.doi.org/10.1145/2814228.2814229

[4] Duc Cuong Nguyen, Dominik Wermke, Yasemin Acar, Michael Backes, Charles Weir, and Sascha Fahl. 2017. A Stitch in Time: Supporting Android Developers in WritingSecure Code. In Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security (CCS ’17). ACM, New York, NY, USA, 1065-1077. DOI: https://doi.org/10.1145/3133956.3133977

[5] Y. Acar, M. Backes, S. Fahl, D. Kim, M. L. Mazurek and C. Stransky, ”You Get Where You’re Looking for: The Impact of Information Sources on Code Security,” 2016 IEEE Symposium on Security and Privacy (SP), San Jose, CA, 2016, pp. 289-305.

[6] Hironori Imai, Akira Kanaoka: Time Series Analysis of Copy&Paste Impact on Android Application Security, 13th Asia Joint Conference on Information Security (AsiaJCIS 2018), 2018

[7] Hironori Imai, Akira Kanaoka: Chronological Analysis of Source Code Reuse Impact on Android Application Security, Journal of Information Processing, Vol.27, pp.603-612, 2019

[8] Li Y., Zhang Y., Li J., Gu D. (2014) iCryptoTracer: Dynamic Analysis on Misuse of Cryptography Functions in iOS Applications. In: Au M.H., Carminati B., Kuo CC.J. (eds) Network and System Security. NSS 2015. Lecture Notes in Computer Science, vol 8792. Springer, Cham

[9] 角田 大輔, 松浦 幹太. 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. コンピュータセキュリティシンポジウム 2019 論文集,2019,1130-1134 (2019-10-14).

[10] Android Developers, ”API reference”, https://developer.android.com/reference?hl=ja, （参照 2020-01-08）

[11] Google LLC, ”google/tink”, https://github.com/google/tink, （参照 2020-01-08）

[12] Facebook, ”Conceal”, https://facebook.github.io/conceal/, （参照 2020-01-08）

[13] Bradley Reaves, Nolen Scaife, Adam Bates, Patrick Traynor, Kevin R.B. Butler, University of Florida. Mo(bile) Money, Mo(bile) Problems: Analysis of Branchless Banking Applications in the Developing World. *24th USENIX Security Symposium*, https://www.usenix.org/node/190885, 2015

[14] Université du Luxembourg, ”AndroZoo”, https://androzoo.uni.lu/, （参照 2020-01-08）