Vol.2013-DPS-154 No.23
Vol.2013-CSEC-60 No.23
2013/3/14

Android アプリケーションにおける暗号処理検証技術の実装及び評価

山本 匠[†]　　河内 清人[†]　桜井 鐘治[†]

Android 端末上で動作するアプリケーション（アプリ）は開発環境が無償で公開されているため，誰でも自由にアプリを作成できる反面，アプリが安全に情報を取り扱っているかが懸念される．そのためユーザは，重要な情報がアプリの中で本当に正しく管理されているかを，アプリのインストール前に確認したいと考えられる．現在までのところ，様々なアプリ検証技術が提案されてはいるものの，アプリのパーミッションを確認したり，重要な情報がネットワーク上に流れ出るかを確認したりするだけであり，重要な情報を適切に保護した上で出力するアプリもリスクの高いアプリとして判定してしまう恐れがある．そこで著者らは既に，アプリが重要な情報を適切に暗号化した上で出力しているかをプログラムの静的解析を用いて検証する方式を提案している．本稿では，同方式のプロトタイプを実装し，検証精度及び検証速度について簡単な評価を行う．

An implementation and evaluation of verification method for appropriateness of encrypting processes in Android Applications

TAKUMI YAMAMOTO[†]　KIYOTO KAWAUCHI[†]
SHOJI SAKURAI[†]

A SDK of Android applications is freely available on the Internet. Although this greatly helps us to develop Android applications, security of an Android application is serious problem. And a user wants to confirm if an application manages important information properly before installs it. So far several techniques for checking security of android applications have been proposed. But they may falsely judge a good application created in a secure manner (e.g., encrypts important information appropriately before the information is outputted from the application, and manages the key for the encryption/decryption safely), as an application may have risk of information leakage, since they just check the types of permissions the application uses or whether important information flows out of the application. Therefore, we have proposed a concept of verification method that checks whether important information is outputted with appropriate encryption process, using static program analysis. In this paper, we implement a prototype system of proposed method and evaluate the performance of the system.

1. はじめに

近年，スマートフォンが急速に普及している．スマートフォンとは，携帯電話端末の機能に加え，パソコン（PC）が持っているような高度な情報処理機能が備わった携帯電話端末である[1]．従来の携帯電話と異なり，非常に多くのアプリが公開されており，ユーザはスマートフォンに好きなアプリをインストールし利用することができる．

総務省が開催するスマートフォン・クラウドセキュリティ研究会の報告[1]によれば，2011 年 12 月末時点で，国内におけるスマートフォンの契約数の OS 別シェアは，Android OS（以降 Android と略記する）が 58.1%，iOS が 37.2%となっており，Android が最も普及していることがわかる．

また，スマートフォンを業務に導入する企業も増加している．ジーエフケーマーケティングサービスジャパンは，2011 年 10 月に国内企業約 1,500 社の IT 関連業務従事者に実施した企業調査より，約 16%の企業がスマートフォンを業務に導入しており約 1 年前の調査と比べ 8%も増加していることを報告している[2]．その中で，スマートフォンの OS 別シェアは Android が 44%と企業利用においても最多となっている．

ここで，Android とは，スマートフォンやタブレット PC などの携帯情報端末を主なターゲットとして Google 社によって開発されたプラットフォームであり，ソースコードやアプリの開発環境が無償で公開されている[3]．Android は，多くのコミュニティが自由にアプリを作成・公開することができ，ユーザも様々なアプリを利用することができるため，人気が高いと考えられる．

一方で，誰でもアプリを開発し公開することができるため，Android アプリの信頼性に対して，疑問を投げかけられてきた．例えば，Android アプリから SD カードにデータを書き込む場合，アプリ側で MODE_PRIVATE（ファイルを作成したアプリのみがアクセス可能なモード）を指定しても，他アプリからそのデータにアクセスできてしまう[4]．このような Android のセキュリティについて知識が乏しい開発者が作成したアプリを利用することは情報漏洩のリスクが高い．

携帯端末のセキュリティ改善活動を行っている OWASP Mobile Security Project では，安全でない記憶領域（Insecure Data Storage）や機密情報の漏洩（Sensitive Information Disclosure）が，携帯端末におけるリスクのトップ 10 に挙

† 三菱電機株式会社
Mitsubishi Electric Corporation

ⓒ 2013 Information Processing Society of Japan

げられており[5]，情報漏洩に関して Android アプリの信頼性を検証する技術のニーズが高まっているといえる．

Android アプリの信頼性を検証する技術は既に，様々な方式が提案されている[6‐9]．Android OS に手を加え動的にアプリの挙動を監視する方式[6]やアプリの設定情報やプログラムコードを静的に解析することで，情報漏洩の可能性を検証する方式[7‐9]がある．

しかし既存研究では，アプリのパーミッション（アプリがユーザに要求する端末のリソースへのアクセス権）の組み合わせを確認するだけであったり，個人情報などの重要な情報がネットワーク上に流れ出るかを確認するだけであったりするため，重要な情報を適切に保護しているアプリ（例えば，個人情報をファイルに保存する前に暗号化している）も情報漏洩のリスクを持ったアプリとして判定してしまう恐れがある．安全なアプリ作成に努めている開発者にとっては，自分のアプリが信頼性の低いアプリと判定されることは望ましいことではない．

仮に，アプリ外部へ情報を出力する前に暗号化が行われていることを検証できたとしても，暗号化に利用した鍵の管理状況について適切に検証しなければ，本来「信頼性の低いアプリ」と判定すべきアプリ（例えば，暗号化に利用された鍵がアプリ内にハードコーディングされているようなアプリ）も，安全なアプリとして誤判定してしまう．

このような課題に対応するために，著者らは既に，Android アプリケーションにおける暗号処理検証技術を提案している[10]．詳細は 3 章で述べるが，提案方式は，個人情報などの重要な情報が暗号化されずにアプリ外部に出力されることを,データフローを追跡することで確認する．さらに，暗号化された情報がアプリ外部に出力される場合には，暗号化に利用された鍵が，外部入力から得られた情報を元に生成されているかを確認することで，適切に暗号化が行われていたかについても検証する特徴を有する．文献[10]では，著者らは提案方式のコンセプトについてのみ報告している．本稿では，提案方式のプロトタイプを実装し，検証精度及び検証速度について簡単な評価を行う．

以下，2 章で Android アプリの信頼性検証技術の既存研究を紹介し，3 章で提案方式の詳細について述べる．4 章で自作のアプリを用いた簡単な評価を行い，5 章で本稿をまとめる．

2. 関連研究

Android アプリの信頼性検証技術の既存研究としては文献[6‐9]が知られている．そのうち，文献[6]は動的解析，残りが静的解析に分類される．

ここで動的解析とは，実際に検査対象のプログラムを実行しその振る舞いを解析する手法である．一方静的解析とは，プログラムを実行せず，コードを解析することでプログラムの特徴を解析する手法である．以下に既存研究について簡単に紹介する．

2.1 動的解析による信頼性検証

Android アプリを動的に解析し信頼性を検証する技術として，文献[6]が知られている．文献[6]では，プログラム内の個人情報などの特定の情報に対してラベル付けを行い（ラベル付けされた情報を汚染データと呼ぶ),汚染データがネットワーク上に流れ出るか否かをリアルタイムに検証する．

本方式では，プログラム内の情報をリアルタイムに監視するため，アプリ実行時のオーバーヘッドが問題となる．また検査対象アプリの実行時のメモリを解析するため，Android OS そのものを修正する必要がある．

2.2 静的解析による信頼性検証

Android アプリを静的に解析し信頼性を検証する技術として，文献[7‐9]が知られている．文献[7]は，アプリがユーザに要求するパーミッションの中に,「インターネットアクセス」と「連絡先データの読み取り」などといった危険な組み合わせが含まれているかを解析する．しかし，あくまでもアプリがそれらのパーミッションを必要としていることまでしかわからず，例えば，連絡先データをインターネットに送信しているかどうかまではわからない．そのため，誤検知が発生しやすい．

文献[8]は，アプリのプログラムコードを静的にデータフロー解析し，個人情報などを取得する関数の戻り値が，通信やファイル出力用の関数の引数に含まれていないかを解析する．たとえ「インターネットアクセス」と「連絡先データの読み取り」のパーミッションが要求されていたとしても，連絡先データが通信用関数の入力に流れていなければ警告は出さないため，文献[7]の方式よりも誤検知が少なくなる．

また，静的解析によりユーザが望まない機能をプログラム内から特定し，その機能が無効になるようアプリを改修する研究も提案されている[9].

2.3 既存研究の課題

Android アプリの信頼性検証技術の既存研究[6-9]では，データの流れやユーザに要求するパーミッションを確認することで，プログラムの信頼性を検証していた．実際に図 1 のような，電話番号を取得しそれをネットワーク経由で送信するような情報漏洩のリスクを持ったアプリを検出することができる．

しかし，重要な情報を適切に保護しているアプリ（例えば，個人情報をファイルに保存する前に暗号化している図 2）も情報漏洩のリスクを持ったアプリとして判定してしまう恐れがある．なぜなら，既存研究ではアプリの中で扱う情報が暗号化されているか否かまでは考慮しないため

である．

仮に，アプリ外部へ情報を出力する前に暗号化が行われていることを検証できたとしても，暗号化に利用した鍵の管理状況について適切に検証しなければ，本来「信頼性の低いアプリ」と判定すべきアプリ（例えば，暗号化に利用された鍵がアプリ内にハードコーディングされているようなアプリ）も，安全なアプリとして誤判定してしまうだろう．この場合，ユーザが入力したパスワードが MD5 や SHA256 などの暗号学的ハッシュ関数（以降　ハッシュ関数と略記する）の入力になっており，ハッシュ関数の戻り値が暗号鍵となっていれば，適切に暗号化されていると考えてもよい．

図 1　既存研究[9]による
情報漏洩のリスクを持ったアプリの検出例

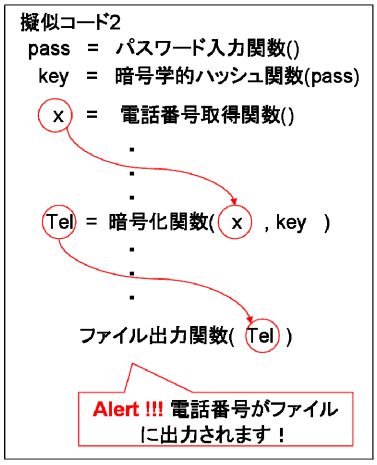

図 2　既存研究[9]による適切なアプリの誤検知の例

3. 提案方式

2 章で述べた課題を解決するために，本章では，重要な情報が適切な暗号化方法を経ないで，アプリ外部に流出することを，プログラムの静的解析を用いて検証する方式を提案する．

本稿では，パスワードなどユーザから入力された情報が MD5 や SHA256 などのハッシュ関数の入力になっており，そのハッシュ関数の出力が鍵として暗号化に利用されている場合に，適切に暗号化されていると定義する．

3.1 コンセプト

提案方式の概観を図 3 に示す．提案方式は大きく分けて以下の 3 つの機能から構成されている．

(1) 前処理機能
(2) データフロー解析機能
(3) 判定機能

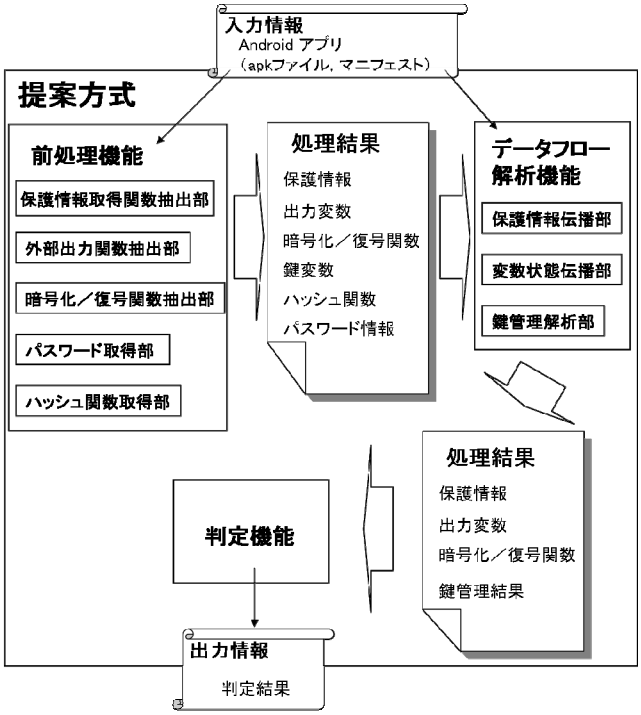

図 3　提案方式の概観

前処理機能では，検査対象の Android アプリ（apk ファイルやマニフェストファイル）を入力として受け取り，プログラム内から保護情報，変数状態，出力変数，暗号化／復号関数，鍵変数，ハッシュ関数及びパスワード情報を取得する．

ここで，保護情報とは個人情報などの重要情報が格納されている変数である．前処理機能の段階では，特定の関数の戻り値が代入された変数のみ保護情報となる．

変数状態とは，プログラム内の各変数に格納されている情報が「平文」か「暗号文」かを表す．前処理機能の段階では，特定の関数の戻り値が代入された変数のみ変数状態が決められる．

出力変数とは，外部出力関数に入力されている変数を示す．暗号化／復号関数はプログラム内の暗号化／復号関数を示す．鍵変数は暗号化／復号関数に鍵として入力されている変数である．ハッシュ関数とはプログラム内のハッシュ関数を示す．パスワード情報とはユーザから入力されるパスワードが格納される変数を示す．

ⓒ 2013 Information Processing Society of Japan

データフロー解析機能は前処理機能で得られた情報を起点にプログラムのデータフローを解析し，プログラム内の他のどの変数が保護情報で,他の変数の状態は何であり，暗号化に利用された鍵が外部入力から得られた情報を元に生成されているかを調査する．

判定機能では，データフロー解析機能で得られた情報を元に，検査対象のアプリが信頼できるかどうかを判定し，判定結果をユーザに出力する．次節では，各機能について詳述する．

3.2 提案方式の各機能

3.2.1 前処理機能

前処理機能は，保護情報取得部，出力変数取得部，暗号化／復号関数取得部，ハッシュ関数取得部及びパスワード取得部から構成されている．

(a) 保護情報取得部

保護情報取得部では，個人情報や機密情報などの保護すべき情報を返す関数をアプリの中から抽出しその関数の戻り値が代入される変数を保護情報とする．保護情報とした変数の状態を「平文」とする．それ以外は「未定」とする．

表 1 保護情報取得用の関数例
パッケージ名とメソッドの引数は省略

クラス#メソッド	関数の説明
TelephonyManager #getLine1Number	端末の電話番号を取得
Account#toString	端末のメールアドレスを取得
Location#getLatitude	位置情報を取得

抽出する関数は，Android で API として公開されている関数から選択される．例えば表 1 のようなものがある．

(b) 出力変数取得部

出力変数取得部では，ファイル出力関数，通信関数及び他アプリに情報を送信する関数をアプリの中から抽出し，その関数の出力用の実引数を出力変数とする．

(a)と同様，抽出する関数は，Android で API として公開されている関数から選択される．例えば表 2 のようなものがある．

表 2 出力用の関数例
パッケージ名とメソッドの引数は省略

クラス#メソッド	関数の説明
Intent#putExtra	他のアプリに情報を出力
OutputStream#write	ファイルに情報を出力
HttpClient#execute	Http 通信で情報を送信

なお，SSL など入力された情報を暗号化して出力するタイプの出力関数については，出力変数は「暗号化出力」とする．それ以外は「平文出力」とする．このような関数としては，例えば表 3 のようなものがある．

(c) 暗号化／復号関数取得部

暗号化／復号関数取得部では，暗号化関数及び復号関数をアプリの中から取得する．さらにその関数の鍵用の実引数を「鍵変数」とする．「鍵変数」は保護情報とする．

また暗号化関数の戻り値が代入される変数の状態を「暗号文」,復号関数の戻り値が代入される変数の状態を「平文」とする．

(a)と同様，取得する関数は，Android で API として公開されている関数から選択される．例えば表 4 のようなものがある．

なお，暗号化／復号関数の戻り値が代入される変数や鍵変数が，プログラム内のどの暗号化／復号関数と対応しているか記録しておく．

表 3 暗号化して出力する関数例
パッケージ名とメソッドの引数は省略

クラス#メソッド	関数の説明
SSLSocket# getOutputStream().write	入力される情報を暗号化して Socket に書き込む
CipherOutputStream #write	入力される情報を暗号化してファイルに書き込む

表 4 暗号化／復号関数の例

クラス#メソッド	関数の説明
Cipher#doFinal ☆	暗号化又は復号を行う

(d) ハッシュ関数取得部

ハッシュ関数取得部では，MD5 や SHA256 などのいわゆる暗号学的ハッシュ関数をアプリの中から取得する．(a)と同様，取得する関数は，Android で API として公開されている関数から選択される．例えば表 5 のようなものがある．

表 5 ハッシュ関数の例

クラス#メソッド	関数の説明
MessageDigest#update	ハッシュ関数にハッシュ対象の情報を入力
MessageDigest#digest	ハッシュ値を出力

☆ あらかじめ，Cipher.ENCRYPT_MODE を指定して Cipher#init を実行すると暗号モード，Cipher.DECRYPT_MODE を指定すると復号モードに切り替わる．

ⓒ 2013 Information Processing Society of Japan

(e)　パスワード取得部

パスワード取得部では，ユーザからの入力を受け取る関数を，アプリの中から抽出し，その戻り値が代入される変数を「パスワード情報」とする．パスワード情報は保護情報とし，「平文」の状態とする．(a)と同様，抽出する関数は，Android で API として公開されている関数から選択される．例えば表 6 のようなものがある．

3.2.2 データフロー解析機能

データフロー解析機能は，保護情報伝播部，変数状態伝播部，鍵管理解析部から構成されている．データフロー解析機能は，前処理機能で得られた情報を元に動作する．

ここでデータフロー解析とは，プログラムのデータの流れを追跡する手法のことをいう．詳細については文献[11,12]を参考にされたい．

表 6　ユーザからの入力を受け取る関数の例

クラス#メソッド	関数の説明
EditView#getText☆	ユーザの入力を取得

(a)　保護情報伝播部

保護情報伝播部は前処理機能で保護対象とした変数に代入されている値が，プログラム内の他のどの変数に伝播していくのかをデータフロー解析をすることで把握する．保護情報の伝播について図 4 の擬似コード 3 を用いて簡単に説明する．

```
a = getTelNo();      // 端末の電話番号取得関数
b = "tel:";
c = b + a;
write(a);            // ファイル出力関数
```

図 4　擬似コード 3

表 7　プログラムの各行での保護情報リスト

行	保護情報のリスト
1	a
2	a
3	a, c
4	a, c

擬似コード 3 の 1 行目で，変数 a に端末の電話番号取得関数の戻り値が代入されている．前処理によって変数 a が保護情報とされたと仮定する．プログラムの 1 行目から順番に処理を確認していき，各処理によってどのように保護情報が伝播していくのかを解析する．

表 7 にプログラムの各行に到達する保護情報のリストを示す．まず，プログラムの 1 行目では，変数 a のみが保護情報として管理される．2 行目には，1 行目の保護情報が到達するが，保護情報を伝播する処理は行われない．3 行目では，変数 a と変数 b を連接した結果を変数 c に代入している．変数 a は保護情報であるため，変数 c も保護情報として管理される．4 行目ではファイル出力関数の実引数に保護情報である変数 c が設定されていることがわかる．

Android の組み込み関数を使った処理については，あらかじめその関数の入力と出力の関係を定義しておき，実引数に保護情報の変数が設定されていた場合に，その関数の戻り値が代入される変数も保護情報とするかどうかを決定する．

例えば図 5 の擬似コード 4 では，2 行目の toUpperCase() 関数によって，入力となる呼び出し元の文字列オブジェクトが大文字に変換される．このような組み込み関数の場合，入力に保護情報の変数が設定されていれば，出力も保護情報とするよう定義しておく．

```
a = getName();   // 端末所有者の名前取得関数
b = a.toUpperCase();   // 大文字に変換する関数
c = "name:";
d = c + b;
write(d);              // ファイル出力関数
```

図 5　擬似コード 4

(b)　変数状態伝播部

変数状態伝播部は前処理機能で「平文」又は「暗号文」と設定された変数の状態を，データフロー解析することでプログラム内の他の変数にも伝播していく．変数の状態の伝播について図 6 の擬似コード 5 を用いて簡単に説明する．

擬似コード 5 の 2 行目で，変数 a に端末所有者の名前取得関数の戻り値が代入されている．前処理によって変数 a が「平文」の状態の保護情報とされたと仮定する．

また enc と dec はそれぞれ暗号化関数と復号関数であり，第 2 引数（k）を鍵とする．変数 d と変数 e には関数 enc と関数 dec の戻り値がそれぞれ代入されており，前処理によって変数 k．変数 d 及び変数 e はそれぞれ「平文」．「暗号文」及び「平文」の状態とされたと仮定する．

```
k = "abcdefg";
a = getName();   // 端末所有者の名前取得関数
b = a;
c = enc(b,k);      // 暗号関数
d = dec(c,k);      // 復号関数
e = c;
f = c + d;
write(c)
```

図 6　擬似コード 5

☆ EditText#setInputType に InputType.TYPE_TEXT_VARIATION_PASSWORD が設定されている場合，ユーザに表示されるダイアログボックス内の入力文字列が伏せ字になるため，パスワードであることがわかる．

ⓒ 2013 Information Processing Society of Japan

プログラムの 1 行目から順番に処理を確認していき，各行の処理によってどのように変数の状態が伝播していくのかを解析する．表 8 にプログラムの各行に到達する変数状態のリストを示す．

1 行目では前処理で得られた変数の状態が保持される．3 行目では「平文」の状態の変数 a が変数 b に代入されているため，変数 b の状態も「平文」となる．6 行目では「暗号文」の状態の変数 c が変数 e に代入されているため，変数 e の状態も「暗号文」となる．7 行目では「暗号文」の状態の変数 c と「平文」の状態の変数 d を連接した結果が変数 f に代入されているため，変数 f も「平文」の状態とする．

Android の組み込み関数を使った処理については，保護情報伝播部と同様に，あらかじめその関数の入力と出力の関係を定義しておき，実引数に設定されている変数の状態（「平文」又は「暗号文」）に応じて，戻り値が代入される変数の状態を決定する．

表 8　プログラムの各行での変数状態リスト

行	変数状態リスト(変数名,状態)
1	(a,平文) (c,暗号文) (d,平文) (k,平文)
2	(a,平文) (c,暗号文) (d,平文) (k,平文)
3	(a,平文) (b,平文) (c,暗号文) (d,平文) (k,平文)
4	(a,平文) (b,平文) (c,暗号文) (d,平文) (k,平文)
5	(a,平文) (b,平文) (c,暗号文) (d,平文) (k,平文)
6	(a,平文) (b,平文) (c,暗号文) (d,平文) (e, 暗号文) (k,平文)
7	(a,平文) (b,平文) (c,暗号文) (d,平文) (e, 暗号文) (f, 平文) (k,平文)
8	(a,平文) (b,平文) (c,暗号文) (d,平文) (e, 暗号文) (f, 平文) (k,平文)

(c)　鍵管理解析部

鍵管理解析部は，前処理機能で「鍵変数」と設定された変数に割り当てられている値が，定数ではなく，外部入力から得られた情報を元に生成されているかを調査する．

変数に割り当てられている値が，定数かどうかを確認するために，コンパイラの最適化でよく利用される定数伝播を用いる．詳細については文献[11,12]を参考にされたい．定数伝播によってプログラム内の各変数に対し，それらに代入されている値が「定数」なのか「非定数」なのかといった情報がラベル付けされる．

さらに，前処理機能で抽出したパスワード情報が MD5 や SHA256 などのハッシュ関数の入力になっておりそのハッシュ関数の戻り値が，そのまま又はさらに他の情報と演算（XOR やハッシュ関数）された結果が，鍵変数に割り当てられている場合に，その鍵変数が利用されている暗号化／復号関数を「安全」とする．

図 7 の擬似コード 6 を例に鍵管理解析の手順を説明する．プログラムの 1 行目から順番に定数伝播を行っていく．表 9 にプログラムの各行に到達する変数情報（「定数」又は「非定数」）のリストを示す．

```
r = 12345;                  // 擬似乱数生成期
p = inputPassword();  // パスワード入力
h = md5(p);                 // MD5
k = r xor h
a = getName();       // 端末所有者の名前取得関数
b = enc(a,k);           // 鍵 k で a を暗号化
write(b);                  // b をファイルに出力
```

図 7　擬似コード 6

表 9　プログラムの各行での変数情報リスト

行	変数情報リスト　（変数名,定数/非定数）
1	(r,定数)
2	(r,定数) (p,非定数)
3	(r,定数) (p,非定数) (h,非定数)
4	(r,定数) (p,非定数) (h,非定数) (k,非定数)
5	(a,非定数) (r,定数) (p,非定数) (h,非定数) (k,非定数)
6	(a,非定数) (b,非定数) (r,定数) (p,非定数) (h,非定数) (k,非定数)

表 10　5 行目の鍵変数 k の代入情報

行	処理
4	K = r xor h
3	h = md5(p)
2	p = inputPassword()
1	r = 12345

変数 r には定数が代入されているため，変数 r は「定数」であることがわかる．2 行目では，変数 p にユーザから入力されたパスワードが代入される．ユーザの入力は未知であるため，変数 p は「非定数」となる．3 行目ではハッシュ関数の MD5 に，「非定数」の変数 p が入力されているため，関数の戻り値が代入される変数 h は「非定数」となる．4 行目では，「定数」の変数 r と「非定数」の変数 h を XOR した結果が変数 k に代入されるため，変数 k は「非定数」となる．

ⓒ 2013 Information Processing Society of Japan

さらに表 10 のように，鍵変数 k の代入関係をさかのぼり，鍵変数 k が，ユーザが入力したパスワードをハッシュ関数で処理したものから生成されていることを確認する．

3.2.3 判定機能

データフロー解析機能で得られた情報から，以下のことを検査し，アプリが信頼できるものかどうかをユーザに出力する．

(a) 「平文出力」の出力変数が，保護情報であり，「平文」の状態である

(b) 「平文出力」の出力変数が，保護情報であり，かつ，「暗号文」の状態であり，暗号化に利用した暗号化関数が「安全」ではない

上記に該当する場合，その旨をユーザに通知する．例えば図 6 の擬似コード 5 の場合，「【警告】: 端末所有者の名前が，暗号化されてファイルに出力されますが，適切に暗号化されていません.」という内容がユーザに通知される．一方，図 7 の擬似コード 6 のように適切に暗号化を行っている場合，「端末所有者の名前が，暗号化されてファイルに出力されますが，適切に暗号化されています.」という内容がユーザに通知される．

4. 評価

提案方式のプロトタイプを実装し評価を行った．

4.1 評価環境

表 11 及び表 12 に示す環境で，提案方式のプロトタイプを実行し評価を行った．

表 11　ハードウェア構成

CPU	インテル Core(TM) i3 CPU 540 @ 3.07 GHz
メモリ	3.24 GB
HDD	149 GB

表 12　ソフトウェア構成

オペレーティングシステム		Windows XP SP3
必要なライブラリ	ログ出力ライブラリ	log4j 1.2.16
	Java 静的解析ライブラリ	soot 2.4.0
	XML ライブラリ	xerces2 2.10.0
Java ランタイム		jre 1.7.0_11

4.2 評価手順

以下の機能を持ったアプリを自作し，提案方式における機密情報の解析機能及び解析時間について評価を行った．今回作成したアプリは 4 種類（A，B，C，D）である．いずれも，2 つの Activity☆ α，β から構成される．Activity α は，電話帳の情報を取得し，その情報を Activity β に Intent☆☆経由で渡す．Activity β は Intent に含まれている電話帳の情報を取得し，取得した情報を特定の URL に送る．

機密情報の状態（暗号文/平文）及び暗号化に利用された鍵について，提案方式のプロトタイプが適切に解析することができるかを確認するために，アプリの種類ごと，以下のように Activity α の処理を変更した．

A：電話帳の情報を平文のまま Activity β に渡す
B：電話帳の情報を固定値で暗号化して Activity β に渡す
C：電話帳の情報を外部入力された情報のハッシュ値で暗号化して Activity β に渡す
D：電話帳の情報を外部入力されたパスワードのハッシュ値で暗号化して Activity β に渡す

暗号化及び外部入力されたパスワードの取得について，図 8 及び図 9 に示すコードでそれぞれ実装した．アプリの実装における API レベルは 13 である．

4.3 評価結果

評価結果を表 13 及び表 14 に示す．表 13 中，「漏洩情報」とは，提案方式によりアプリを解析した結果，特定された外部出力される機密情報である．「暗号化」は，特定された「漏洩情報」が外部出力されるときに暗号化されているかどうかを示す．「鍵」は，「漏洩情報」の暗号化に利用された鍵の管理情報を示す．「判定」は，解析した結果よりユーザに警告を出すかどうかを示す．表 14 中，「実ステップ数」とは，アプリのプログラムのステップ数を示す．「解析時間」とは解析に要した時間を示す．

4.4 考察

4.3 節の評価結果より，今回自作したアプリにおいては，提案方式は，アプリが重要な情報を適切に暗号化した上で出力しているかどうかを解析することができた．

解析時間については，100～200 ステップ程度の小規模なアプリに対して，5～6 秒も要してしまっていることが見てとれる．現状のデータフロー解析では，解析対象のプログラムのデータフローが収束するまで繰り返し解析を行う Chaotic Iteration [12]と呼ばれるアルゴリズムを用いており，効率が悪い．より効率的なアルゴリズムとしては Reps らが提案するアルゴリズム[13]が知られているが，解析に扱うデータフローに特別な制約（distributive 性）を必要とする．提案方式に Reps らの解析アルゴリズムを導入可能かについて今後検討していく予定である．

☆ Activity とは，Android において画面 1 つを表現するオブジェクトである．

☆☆ Intent とは，Android において，Activity やアプリ間の情報の受け渡しに利用されるオブジェクトである．

ⓒ 2013 Information Processing Society of Japan

表 13　判定結果

アプリ	漏洩情報	暗号化	鍵	判定
A	電話帳	無	無	警告有 3.2.3 の(a)に該当
B	電話帳	有	ハッシュされた定数	警告有 3.2.3 の(b)に該当
C	電話帳	有	ハッシュされた外部入力	警告有 3.2.3 の(b)に該当
D	電話帳	有	ハッシュされたパスワード	警告無

表 14　アプリのサイズと解析時間

アプリ	実ステップ数（Step）	解析時間(ミリ秒)
A	154	4687
B	170	5375
C	180	5062
D	181	4969

```
byte[] b_key = key.getBytes();
byte[] b_plain = plain.getBytes();
MessageDigest md = MessageDigest.getInstance("MD5");
byte[] md_key = md.digest(b_key);
SecretKeySpec keySpec = new SecretKeySpec(md_key, "AES");
Cipher cipher = Cipher.getInstance("AES");
cipher.init(Cipher.ENCRYPT_MODE, keySpec);
byte[] b_cipher = cipher.doFinal(b_plain);
String b64_cipher =
            Base64.encodeToString(b_cipher, Base64.DEFAULT);
```

図 8　暗号化処理　文字列変数 plain，key 及び b64_cipher にはそれぞれ，平文，鍵及び暗号文が格納される

```
new AlertDialog.Builder(TelephoneList.this)
 .setIcon(android.R.drawable.ic_dialog_info)
 .setView(editView)
 .setPositiveButton("OK", new DialogInterface.OnClickListener() {
  public void onClick(DialogInterface dialog, int whichButton) {
  editView.setInputType(
   InputType.TYPE_TEXT_VARIATION_PASSWORD);
   pass = editView.getText().toString(); }}).show();
```

図 9　外部入力されたパスワードの取得
文字列変数 pass にパスワードが格納される

5. おわりに

本稿では，著者らが文献[10]で提案した，Android アプリケーションにおける暗号処理検証技術のプロトタイプを実装し，自作のアプリを用いた簡単な評価を行った．今回用いたアプリにおいては，提案方式は，アプリが重要な情報を適切に暗号化した上で出力しているかどうかを解析することができた．今後は，マーケットに公開されているアプリに対して，提案方式を適用し評価を行っていく予定である．

参考文献

1) 総務省，“スマートフォン・クラウドセキュリティ研究会 最終報告 ～スマートフォンを安心して利用するために実施されるべき方策～”，http://www.soumu.go.jp/main_content/000166095.pdf (2013 年 2 月確認).
2) ジーエフケー マーケティングサービス ジャパン株式会社, “企業のスマートフォン，タブレット型端末利用状況調査 企業の活用は拡大”, http://www.gfkrt.com/imperia/md/content/documents/press_release_111115.pdf (2013 年 2 月確認).
3) Android Developers, http://developer.android.com/index.html (2013 年 2 月確認).
4) JSSEC 日本スマートフォンセキュリティ協会，“Android アプリのセキュア設計 セキュアコーディングガイド 【みんなでスマフォが安全に使える世界へ！】”, http://www.jssec.org/dl/android_securecoding.pdf (2013 年 2 月確認).
5) OWASP Mobile Security Project, “Top 10 Mobile Risks, Release Candidate v1.0”, https://www.owasp.org/index.php/OWASP_Mobile_Security_Project (2013 年 2 月確認).
6) W. Enck, P. Gilbert, B-G. Chun, L.P. Cox, J. Jung, P. McDaniel and A.N. Sheth. “An Information Flow Tracking System for Realtime Privacy Monitoring on Smartphones”, In Proc. 9th Usenix Symposium on Operating Systems Design and Implementation, 2010, pp. 393-408.
7) W. Enck, M. Ongtang and P. McDaniel, “On lightweight mobile phone application certification”, In Proc. 16th ACM Conference on Computer and communications security, 2009, pp. 235-245.
8) A.P. Fuchs, A. Chaudhuri and J.S.Foster, “ScanDroid: Automated Security Certification of Android Applications”, http://www.cs.umd.edu/~avik/papers/scandroidascaa.pdf.
9) L. Batyuk, M. Herpich, S.A. Camtepe, K. Raddatz, A-D. Schmidt and S. Albayrak, “Using Static Analysis for Automatic Assessment and Mitigation of Unwanted and Malicious Activities Within Android Applications”, In Proc. 6th International Conference on Malicious and Unwanted Software, 2011, pp.66-72.
10) 山本 匠，河内 清人，桜井 鐘治，“Android アプリケーションにおける暗号処理検証技術の提案”，コンピュータセキュリティシンポジウム 2012（CSS2012）論文集，1B1-4，2012.
11) 中田育男，“コンパイラの構成と最適化”，ISBN4-254-12139-3 C3041，朝倉書店，1999 年.
12) F. Nielson, H. R. Nielson, C. Hankin, “Principles of Program Analysis.”, ISBN 3-540-65410-0, Springer, 2005.
13) T. Reps, S. Horwitz and M. Sagiv, “Precise interprocedural dataflow analysis via graph reachability”, In proceedings of the 22nd ACM SIGPLAN-SIGACT symposium on Principles of programming languages, 1995.

ⓒ 2013 Information Processing Society of Japan