 CMC, vol.56, no.3, pp.447-465, 2018

Provably Secure APK Redevelopment Authorization Scheme in the Standard Model

Daofeng Li [1, 3, *], Mingxing Luo [2], Bowen Zhao[1, 3] and Xiangdong Che [4]

Abstract: The secure issues of APK are very important in Android applications. In order to solve potential secure problems and copyrights issues in redevelopment of APK files, in this paper we propose a new APK redevelopment mechanism (APK-SAN). By exploring sanitizable signature technology, APK-SAN allows the original developer to authorize specified modifier who can redevelop the designated source code of APK files. Our scheme does not require interactions between the developer and modifiers. It can reduce the communication overhead and computational overhead for developers. Especially, the signature of redeveloped APK files is valid and maintains the copyrights. The proposed APK-SAN signature can effectively protect the security of the redeveloped APK files and copyrights of the developer and modifier.

Keywords: Sanitizable signature, APK signature mechanism, redevelopment, ID-based signature scheme.

1 Introduction

Due to the popularity and open source characteristics of Android system, various Android applications require to be redeveloped in different institutions, regional, and groups, such as alipay, health monitoring, weather forecast [Qin (2016); Enck, Damien and McDaniel (2011); Chen, Ying, Jiao et al. (2014); Qu, Keeney, Robitzsch et al. (2016)]. In these applications, one company Bob can require one developer Alice to redevelop his APP for special requirements. Alice authorizes Bob to redevelop specified part of APK files (one of Android files) [Park and Choi (2015); Jeon, Micinski and Vaughan (2012); Neidhardt (2011); Kim, Yoon, Yi et al. (2012); Karim, Kagdi and Penta (2016)]. In procedure of redeveloping, Bob should finish tasks accord to the authorization. If Alice's authorization is not satisfied, the unspecified files of Bob are over-privileges [Toorani and Beheshti (2008); Silva, Amcrim and Ribeiro (2015); Pearce, Nunez and Wagner (2012); Choi, Sung, Choi et al. (2015); Barrera, Kayacik, van Oorschot et al. (2010); Blasing, Batyuk,

[1] School of Computer, Electrical and information, Guang Xi University, Nanning, 530004, China.

[2] Information Security and National Computing Grid Laboratory, Southwest Jiaotong University, Chengdu, 610031, China.

[3] Guangxi Colleges and Universities Key Laboratory of Multimedia Communications and Information Processing, Guangxi University, Nanning, 530004, China.

[4] School of Information Security & Applied Computing, College of Technology, Eastern Michigan University, Michigan, 48197, USA.

* Corresponding Author: Daofeng Li. Email: ldf_0123@163.com.

Schmidt et al. (2010); Felt, Wang, Moshchuk et al. (2011); Egele, Brumley, Fratantonio et al. (2013)]. This problem is very important and relates to copyrights protection.

The over-privilege is a serious problem in various mobile applications. It can be caused by malicious code embedding or data manipulating. It is an unavoidable problem derived from interactions in applications. In order to protect the legitimacy, copyrights and permission of Android APKs, there are various mechanisms deployed in Android system, such as signature [Toorani and Beheshti (2008); Egele, Brumley, Fratantonio et al. (2013); Shao (2014); Ma, Li and Deng (2016)], sandbox [Blasing, Batyuk, Schmidt et al. (2010)] and permission management [Karim, Kagdi and Penta (2016); Barrera, Kayacik, van Oorschot et al. (2010); Felt, Wang, Moshchuk et al. (2011)]. Signature is required by the APK developer to protect the integrity of APK before publishing APK application in APP store. Specifically, the developer uses the self-signed digital certificate for the compilation of packaging APK. The signature ensures the integrity, authenticity, non-repudiation of APK, and the uniqueness of APK providers, further to protect the developer's copyrights. Sandbox requires that all applications have to run on the Dalvik virtual machine. Sandbox and POSIX realize automatic management of Android applications that can ensure secure operations of APK. Copyrights management mainly protects commercial applications and Android system.

Authorization permissions of Android applications are divided into four levels: normal, dangerous, signature, and signature or system. These permissions are defined and declared in Android manifest file by the developer. However, it is coarse-grained and cannot protect users' sensitive information for some Android applications [Silva, Amcrim and Ribeiro (2015); Choi, Sung, Choi et al. (2015)]. AdDroid is proposed [Pearce, Nunez and Wagner (2012)]. It is an Android advertising privilege separation mechanism, and used to protect the users' privacy information. Meanwhile, a practical and efficient permission certification technique is explored [Dimitriadis, Efraimidis and Katos (2016)] by using the runtime information and static analysis to solve the problems of unauthorized APK.

In order to prevent embedded malicious code and copyrights issues, the third party needs to be authorized for modifying APK file. In this paper, we focus on licensing source codes for modifying. We propose a provably secure APK authorization scheme that is named as APK-SAN authorization mechanism in a standard model [Ateniese, Chou, DeMedeiros et al. (2005); Brzuska, Fischlin, Freudenreich et al. (2009)]. Our authorization has unique attribute, i.e. identity-based sanitizable signature. It allows APK developer to authorize the third party (the modifier) who can redevelop and modify APK licensing area or position. Moreover, the signature of new APK is still valid. The new mechanism is useful for protecting the copyrights of the modified APK, and can prevent from flooding malicious APK files. It does not require interacting with each other after authorizing the third party. This feature provides a fine-grained authorization. To sum up, APK-SAN authorization mechanism has the following improvements:

(1) Fine-grained authorization. When the third party applies for redeveloping a native APK, the developer authorizes he/she to modify files and position. The developer takes use of the identity-based sanitizable signature. If the third party complies with the permission license and redevelops the designated file or position, and the

modified APK is legal and valid. Otherwise, the signature verification of the modified APK can detect an unauthorized redevelopment.

(2) Provable security. APK-SAN authorization mechanism mainly adopts the identity-based sanitizable signature to implement the access authorization. In this paper, we prove our scheme under a standard model.

The rest of the paper is organized as follows. In Section 2, we provide an overview of the An-droid signature mechanism, identity-based sanitizable signature scheme, and secure model. We present a protection mechanism using the identity-based sanitizable signature scheme and over-privileged applications in Section 3. In Section 4, we evaluate the security and performance of the proposed scheme while we conclude the paper in the last section.

2 Preliminaries

2.1 Android native signature mechanism

Unsigned APK usually makes up of res folder, lib folder, resources, files (rousouces.arsc, classes.dex and AndroidManifest.XML), as shown in Fig. 1.

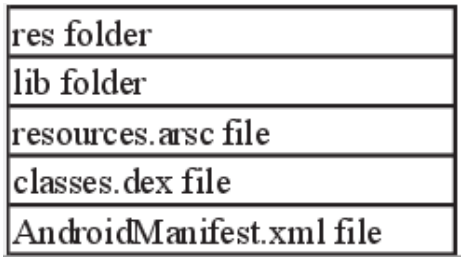

Figure 1: Unsigned APK file format

When the third party redevelops APK, these files (folders) need to be modified. The developer of the original APK has to specify which files can be modified when he authorizes the redeveloper. The resources of each file (folders) are as follows:

(1) res folder: Deposits resource files, such as pictures, layout files, etc.;

(2) lib folder: Deposits.so dynamic link library, and .so dynamic link library are executable files, which are generated by the native code of JNI layer compiling and linking and performed by the Linux kernel. If there are no native codes in the source codes, no lib folder is included in APK;

(3) resources.arsc file. Compiled binary resource file;

(4) classes.dex file. Dalvik executable file, performed by Dalvik virtual machine. Android compiler compiles the project source codes;

(5) .class files. DX tool converts all .class files to executable classes.dex file; AndroidManifest.xml file: Global configuration file.

Android system specifies that all published APK files have to be signed by developers prior to install. By signing the compiled and packaged APK files, one can ensure the integrity of APK, authenticity and non-repudiation, while the rights and interests of developers.

APK signature is as follows:

(1) Generate MANIFEST.MF file. Traverse all files in APK file. Use SHA-1 hash algorithm to generate hash value of files, and encode each hash value with Base64 encoding. All of them are saved in MANIFEST.MF file.

(2) Generate CERT.SF file. Compute MANIFEST.MF file and its content hash value with SHA-1 hash algorithm. Encode each hash value with Base64 encoding, and then store them in CERT.SF file.

(3) Generate CERT.RSA file. Sign CERT.SF file and its contents with RSA signature algorithm and the developer's private key. Create a CERT.RSA file including signature and self-signed public key certificate of the developer.

(4) Package all generated signature files. Package all files of MANIFEST.MF, CERT.SF and CERT.RSA in "/META-INF" directory.

2.2 Identity-based sanitizable signature scheme

In sanitizable signature scheme, the signer allows the specified modifier to modify the designated part without interacting with each other, where the modified signature is still valid. In this paper, we propose a new scheme consisting of the signer, sanitizer and verifier using the identity-based sanitizable signature scheme (IBSSS for short). New scheme is implemented as follows:

Let user's identity be $ID = (ID_1, ID_2,\ldots, ID_n) \in \{0,1\}^{\lambda}$ and signature message be $M = (M_1, M_2, \ldots, M_n) \in \{0, 1\}^{\lambda}$. Signer executes the following operations:

(1) **Setup system parameters.** Given a security parameter k, private key generator (PKG) generates system parameters *params* and master key *msk*. *params* is published while *msk* is kept confidentially.

(2) **Create user keys.** Given a user and its identity *ID*, PKG calculates user's private key d_{ID} with master key *msk*. d_{ID} is sent to a user via a secure channel.

(3) **Sign.** System parameters *params*, signer's identity *ID*, message M and signer's private key d_{ID} are input parameters of signature algorithm. Signer outputs signature σ and secret information ψ.

(4) **Sanitize.** System parameters *params*, Sanitizer's identity ID', secret information ψ and signature σ of message M are input parameters of sanitizable algorithm. The sanitizer outputs a new signature σ' of message M'.

(5) **Verify.** System parameters *params*, signer's identity ID_i, messages/signatures (M_i', σ_i') and (M_i, σ_i) are input parameters of the verification algorithm. If the verification is valid, outputs "True". Otherwise, outputs "False".

2.3 Secure model

Diffie-Hellman Problem Let G_1 be an additive group of order q, and G_2 be a multiplicative groups, g be a generator of G_2. The map e: $G_1 \times G_1 \rightarrow G_2$, possess the following properties:

(1) **Bilinearity.** For $\forall a, b \in Z_q$, $P, Q \in G_1$, and $e(Pa, Qb) = e(P, Q)^{ab}$;

(2) **Nondegeneracy.** There exists $P, Q \in G_1$, makes $e(P, Q) \neq 1$;

(3) **Computability.** There exists an efficient algorithm to compute e(P, Q) for P, $Q \in G_1$.

Definition 1. Computational Diffie-Hellman (CDH) Problem. G_1 is a finite additive group. Given g, g^a, $g^b \in G_1$, for unknown $a, b \in Z_q^*$ compute g^{ab}.

Definition 2. The (ε, t)-CDH assumption. If there is no t-time adversary who has at least ε success probability of solving CDH problem for group G_1, then the assumption of CDH problem is correct.

Obviously, if (ε, t)-CDH hypothesis is true, CDH problem is a mathematical puzzle [Brzuska, Fischlin, Freudenreich et al. (2009); Waters (2005)].

Combining with the secure model of sanitizable signature and secure mechanisms provided by APK-SAN authorization mechanism, we present two secure models with the unforgeable and immutable attributes.

Unforgeability: Signature generated by the signer and sanitizer cannot be forged in scheme.

Immutability: Sanitizer can only modify the designated part of files by the signer.

For the unforgeability and immutability, the interactive processes of challenger C and attacker A are as follows:

(1) **Setup system parameters**.

(2) **Request private key.** Attacker A specifies an identity ID for APK developer. Challenger C executes the key generation algorithm to generate identity ID for APK developer or private key d_{ID} for modifier, and sends d_{ID} to A;

(3) **Request sign.** Attacker A specifies an identity for the APK developer or modifier, signatures messages M and signer's private key d_{ID}. Challenger C executes the extraction key and signature algorithms to generate a signature σ of (ID, M), and sends σ to A;

(4) **Forage and modify signature.** Attacker A can successfully modify specifications of an APK developer with the identity ID or signature of the modifier if the following conditions are satisfied: (a) Attacker A does not specifies an APK developer with the identity ID' or request to extract the modifier's private key: (b) Attacker A does not request sign for developed APK (Unforgeability); (c) Attacker A can sign unspecified part of APK (allow to modify), and signature σ' is still valid (Immutability).

If an attacker A successfully forges or modifies an APK signature, it is an instance solving CDH problem.

3 IBSSS based APK authorization mechanism

In order to facilitate the normal operation mechanism, we assume that the developer of a native APK file can authorize modifier to modify APK file if there is a modifier apply to redevelop. Meanwhile, in this paper, "redevelopment" is synonymous with "modification", and the user represents modifier. The fine-grained authorization in APK-SAN is defined as follows:

Definition 3. Fine-grained authorization means that the developer of a native APK file has permissions to authorize the redevelopment or modify the designated APK file.

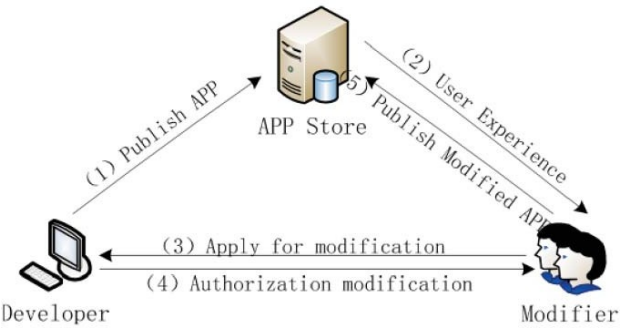

Figure 2: The process of the redevelopment authorization request of APK file. There are five steps to complete a redevelopment. (1) The original developer publishes an APP in APP store. (2) A special user downloads APP and experiences servers. (3) The user or modifier applies a redevelopment of APP for some special goals. (4) The developer authorizes and specifies modifiable files or source codes of APP. (5) Modifier publishes the modified APP

In this section, combing with the identity-based sanitizable signature, we propose a finegrained APK redevelopment authorization mechanism (APK-SAN). APK redevelopment process is shown in Fig. 2. Specific steps are as follows:

(1) **Publish applications.** The developer publishes signed APK file in APP store;

(2) **Users experience.** The users download APK from APP market. According to special requirements, users can request to redevelop APK during the process of experiencing. For example, they find some bugs which may affect users' experience. So, it should be repaired. In another case, users need to add some business functions in the original APK to complete the customization;

(3) **Request authorization.** The user sends the redevelopment request of the native APK authorization;

(4) **Authorization.** The developer authorizes and specifies the user who can modify which files. In this step, we use the identity-based sanitizable signature to complete the authorization. Section 3.2 contributes specific implementation of this component;

(5) **Publish modified applications.** The user validates the developer's authorization legitimacy and effectiveness. Section 3.3 explains the detailed implementation of this component. And then, according to the modifying authorization, user modifies the native APK file, and signs the modified APK file. Finally, user republishes it in APP store. The contents of the specific implementation are detailed in Section 3.4.

The operations of Steps (4) and (5) are used to complete the APK-SAN authorization mechanism that consists of four parts: key generation, APK redevelopment authorization, redevelopment process, and modified validation.

3.1 Key generation

Setup system parameters. Given a security parameter k, PKG chooses two cyclic groups G_1 and G_2 of prime order q, a generator g and an admissible bilinear pairing e: $G_1 \times G_1 \rightarrow G_2$, PKG chooses a random integer $\alpha \in Z_q^*$, computes $g_1=g^{\alpha}$, and selects $g_2 \in G_1$. Furthermore, he chooses $u', m' \in G_1$ and two vectors $\mathbf{u} = (u_1,\ldots,u_n)$, $\mathbf{v}=(v_1,\ldots, v_n)$ of length n, whose entries are random elements in G_1. The system parameters are $params=(G_1, G_2, e, q, g, g_1, g_2, u', m', \mathbf{u}, \mathbf{v})$, and the master key is g_2^{α}.

User's private key. Each developer use identity information $ID_i=Hash(Com_i || IDE_i || t_i)$ as its public key. Here, Com_i is a formal data representation of company's name; IDE_i is unique identification number of the developer; t_i is valid time of the developer's public key. ID_i is sent to PKG. PKG randomly picks r_{ID_i} and computes private key d_{IDi} as follows:

$$\begin{aligned} d_{ID_i} &= (d_{ID_i}^{(1)}, d_{ID_i}^{(2)}) \\ &= (g_2^{\alpha}(u'\prod_{j=1}^{n} u_j^{ID_i^j})^{r_{ID_i}}, g^{r_{ID_i}}) \end{aligned} \tag{1}$$

where $i = 1, 2,..., n$.

3.2 APK file generation and publishing

Before publishing APK, the authorization is that the developer specifies modifiable files (folders) and source codes.

(1) Setup APK file. After completing Android development, the developer packages Java files, resources XML file, Class files, and Dex file into APK file. And then, he/she computes hash values of res folder, lib folder, resources.arsc file, classes.dex file and An- droidManifest.xml file in APK, and encodes them with based64 encoding, which generates Manifest.MF and CERT.SF files as follows:

Manifest.MF file

$H_1=Base64(Hash(d_{des}||res||ID_{des}))$

$H_2=Base64(Hash(d_{des}||lib||ID_{des}))$

$H_3=Base64(Hash(d_{des}||resource.arsc||ID_{des}))$

$H_4=Base64(Hash(d_{des}||classes.dex||ID_{des}))$

$H_5=Base64(Hash(d_{des}||Ardroidmanisfest.xml||ID_{des}))$

CERT.SF file

$h_1=Base64(Hash(d_{des}||H_1||ID_{des}))$

$h_2=Base64(Hash(d_{des}||H_2||ID_{des}))$

$h_3=Base64(Hash(d_{des}||H_3||ID_{des}))$

$h_4=Base64(Hash(d_{des}||H_4||ID_{des}))$

$h_5=Base64(Hash(d_{des}||H_5||ID_{des}))$

(2) Sign APK file. The developer randomly (uniform probability) picks $\overline{r}_{ID_{des}}, r'_{ID_{des}} \in Z_q^*$, and signs APK file using Eq. (2), which generates CERT.SAN file:

$$\begin{aligned}\sigma_1 &= d_{ID_{des}}^{(1)}(v'\prod_{i=1}^{5} v_i^{h_i})^{r'_{ID_{des}}} \\ &= g_2^{\alpha}(u'\prod_{i=1}^{5} u_i^{ID'_{des}})^{\overline{r}_{ID_{des}}}(v'\prod_{i=1}^{5} v_i^{h_i})^{r'_{ID_{des}}}\end{aligned} \tag{2}$$

$$\sigma_2 = d_{ID_{des}}^{(2)} = g^{\overline{r}_{ID_{des}}}, \tag{3}$$

$$\sigma_3 = g^{r'_{ID_{des}}} \tag{4}$$

(3) The developer packages Manifest.MF, CERT.SF and CERT.SAN files into APKSAN.SIG files that are stored in "/META-INF" directory, and publishes in APK file.

3.3 Redevelopment authorization

After publishing APK, if the developer receives the requirement of redevelopment from a user, he completes authorization operations as follows:

(1) The developer specifies modifiable files (folders) and programmable source codes. We assume that a modifiable folders' set $h' = \{h'_i\}, h'_i$ includes the corresponding hash codes, and $h' \subset \{h_i\}, i \in K = \{1,2,...,5\}$.

(2) Let $K' = \{i \in K \mid h_i = h'_i\}, K'' = \{i \in K \mid h_i \neq h'_i\}$ be an index set of the modifiable folders. K'' is an index set of the unmodifiable folders. The developer completes the authorization as follows:

$$\sigma_1^{aut} = \sigma_1, \tag{5}$$

$$\sigma_2^{aut} = \sigma_2(v'\prod_{i?\ '}^{5} v_i^{h''_i})^{r'_{ID_{des}}}, \tag{6}$$

$$\sigma_3^{aut} = \sigma_3(v'\prod_{i?\ ''}^{5} v_i^{h_i})^{r'_{ID_{des}}}, \tag{7}$$

where $h'' = Base64(Hash(ID_{rev} \| h_i \| d_{des}))$ for $i \in K'$. σ_2^{aut} is the authorization of a given modifier with the identity ID_{rev} for an index $i \in K'$. σ_3^{aut} is the immutable folder.

(3) The developer sends the authorized APK file, corresponding source codes and authorization information h', and $\sigma^{aut} = (\sigma_1^{aut}, \sigma_2^{aut}, \sigma_3^{aut})$ to the modifier.

3.4 Redevelopment process

In the process of redevelopment, the modifier firstly verifies the authorization legitimacy and effectiveness. And then, he/she modifies files according to the authorization information, and finally signs the modified APK file. After receiving the authorized APK file and the corresponding source codes from the developer, the modifier retrieves secret authorization information and signature information (v_j^r, σ) from APK AUTH.Sig in APK

file "/META-INF" directory as follows:

(1) Verifying the effectiveness of the signature σ. If the signature is efficient, the modifier executes the Step (2). Otherwise, terminates the protocol.

(2) Combing with the authorization information h', the modifier modifies the modifiable files and the corresponding source codes to generate a new APK file, and reintegrates a new APK file according to the format of Fig. 1 into n shares as $\{h_1^{rev}, h_2^{rev}, ..., h_5^{rev}\}$, where

$$h_i^{rev} = h_i, i \in K'', \tag{8}$$

$$h_i^{rev} = Base64(Hash(d_{rev} \| H_i^{rev} \| ID_{rev})), \ \ i \in K' \tag{9}$$

Here, $H_i^{rev} = Base64(Hash(d_{rev} \| file(i) \| ID_{rev}))$ and $file(i)$ is the modified i-th file (folder).

(3) After completing the redevelopment, using Eq. (4) the modifier randomly chooses $r'_{ID_{des}}, \overline{r}_{ID_{des}} \in Z_q^*$ and calculates the signature of the modified files as:

$$\overline{\sigma} = (\overline{\sigma}_1, \overline{\sigma}_2, \overline{\sigma}_3) \tag{10}$$

where σ_i are defined as $\overline{\sigma}_1 = \sigma_1^{aut}(u'\prod_{i\in K'} u_i^{ID_{rev}^i})^{\overline{r}_{ID_{rev}}}(v'\prod_{i\in K'} v_i^{h_i^{rev}})^{r'_{IDrev}}$, and $\overline{\sigma}_2 = \sigma_2^{aut} g^{r'_{IDrev}}$ and $\overline{\sigma}_3 = \sigma_3^{aut} g^{\overline{r}_{IDrev}}$.

(4) Generating the signature file APKSAN.SIG and storing in the corresponding "/META- INF" directory. The modifier saves the signature $(\sigma, \overline{\sigma})$ into the file, and repackages them to generate a new APK file.

4 Security analysis of APK-SAN

4.1 Validation

The verifier verifies the legitimacy of the signature $(\sigma, \overline{\sigma})$ and σ^{aut}. In this section, we only show the verification of σ_1 and σ_2^{aut} as follows:

$$\begin{aligned}
e(\sigma_1, g) &= e(g_2^{\alpha}(u'\prod_{i=1}^{5} u_i^{ID_{des}^i})^{\overline{r}_{ID_{des}}}(v'\prod_{i=1}^{5} v_i^{h_i^{des}})^{r'_{IDdes}}, g) \\
&= e(g_2^{\alpha}, g_2)\, e(g, g_2^{\alpha}(u'\prod_{i=1}^{5} u_i^{ID_{des}^i})^{\overline{r}_{ID_{des}}}\, e(g, (v'\prod_{i=1}^{5} v_i^{h_i^{des}})^{r'_{IDdes}}) \\
&= e(g_2^{\alpha}, g_2)\, e(g^{\overline{r}_{ID_{des}}}, u'\prod_{i=1}^{5} u_i^{ID_{des}^i})\, e(g^{r'_{IDdes}}, v'\prod_{i=1}^{5} v_i^{h_i^{des}}) \\
&= e(g_1, g_2)\, e(\sigma_2, u'\prod_{i=1}^{5} u_i^{ID_{des}^i})\, e(\sigma_3, v'\prod_{i=1}^{5} v_i^{h_i^{des}}),
\end{aligned}$$

$$\begin{aligned}
e(\sigma_2^{aut}, g) &= e(\sigma_2 (v' \prod_{i \in K''} v_i^{h_i''})^{r'_{ID_{des}}}, g) \\
&= e(\sigma_2, g) e(g^{r'_{ID_{des}}}, v' \prod_{i \in K''} v_i^{h_i''}) \\
&= e(\sigma_2, g) e(\sigma_3, v' \prod_{i \in K''} v_i^{h_i''}),
\end{aligned}$$

$$\begin{aligned}
e(\overline{\sigma}_1, g) &= e(\sigma_1^{aut} (u' \prod_{i \in K'} u_i^{ID^i_{rev}})^{\overline{r}_{ID_{rev}}} (v' \prod_{i \in K'} v_i^{h_i^{rev}})^{r'_{ID_{rev}}}, g) \\
&= e(\sigma_1 (u' \prod_{i \in K'} u_i^{ID^i_{rev}})^{\overline{r}_{ID_{rev}}} (v' \prod_{i \in K'} v_i^{h_i^{rev}})^{r'_{ID_{rev}}}, g) \\
&= e(\sigma_1, g) e(g^{\overline{r}_{ID_{rev}}}, u' \prod_{i \in K'} u_i^{ID^i_{rev}}) e(g^{r'_{ID_{rev}}}, v' \prod_{i \in K'} v_i^{h_i^{rev}}) \\
&= e(\sigma_1, g) e(\sigma_2, u' \prod_{i \in K'} u_i^{ID^i_{rev}}) e(\sigma_3, v' \prod_{i \in K'} v_i^{h_i^{rev}})
\end{aligned}$$

Similarly, one can verify $\overline{\sigma}, \sigma_2^{aut}$, σ and σ_3^{aut}. Therefore, all the signatures are valid. So, the authorization is legitimacy. Meanwhile, the sanitizer can only modify the designated part and position of an APK file.

4.2 Security analysis of the proposed scheme

Definition 4. An adversary A has attack ability $(t, q_e, q_s, \varepsilon)$: In APK-SAN scheme, if A makes q_e private key extraction queries and q_s signature queries, the adversary with a non-negligible advantage ε can successfully forge the original APK signature or the modifier's signature with time t.

Definition 5. If APK-SAN authorization can resist the attack in Definition 4, it is secure for $(t, q_e, q_s, \varepsilon)$.

Theorem 1. If the assumption (ε', t')-CDH is true, APK-SAN authorization is secure for $(t, q_e, q_s, \varepsilon)$.

Proof. Assume that there is an adversary A who is able to successfully attack APK-SAN authorization. It means that A can successfully forge the authorization σ of APK-SAN authorization. Namely, A makes q_e private key extraction queries and q_s signature queries, according to APK-SAN, it can successfully forge validly sanitizable signature with a non-negligible advantage ε and time t. Under these assumptions, given a group G_1 and its generator g, there is an algorithm (Algorithm 1) with advantage ε', which can solve an instance of CDH problem with time t'. It means that after receiving g^a and g^b, A with Algorithm 1 can compute g^{ab}, where $\varepsilon' = 1/(16q(q+q)(n+1)^2)$ and $t' = t + O(5q_e + (2n+4)q_s)$ t_{G_1}. The length of APK file is n bits. It costs time t to execute the exponent arithmetic of the group G_1.

When the attacker A can successfully attack APK-SAN authorization and forge an authorization (a signature), Algorithm 1 can solve a specific CDH problem instance which involves five steps of attack scenario as follows:

Algorithm 1

S1 Assigns system parameters. Denote r_u and r_m as two integers satisfying $0<r_u<q$, $0<r_m<q$. Denote s_u and s_m as integers satisfying $0<s_u<n$, $0<s_m<n$, $r_u(n+1)<q$, and $r_m(n+1)<q$. Let $r_u=2(q_e+q_s)$, $r_m=2q_s$, $x'=\{x_1',x_2',...,x_n'\}\in Z_{r_u}^n$, $y'=\{y_1',y_2',...,y_n'\}\in Z_{r_u}^n$, $z'=\{z_1',z_2',...,z_n'\}\in Z_{r_u}^n$, and $w'=\{w_1',w_2',...,w_n'\}\in Z_{r_u}^n$. The identities of the developers are represented as $ID*=\{ID_1^*,ID_2^*,...,ID_n^*\}$. APK file is constructed according to the developer with its identity $ID*$. Its encoding is represented as $h*=\{h_1^*,h_2^*,...,h_n^*\}$. System parameters $\{g_1,g_2,u',v',\mathrm{u},\mathrm{v}\}$ are constructed as follows:

$$g_1=g^a, \tag{11}$$

$$g_2=g^b, \tag{12}$$

$$u'=g_2^{-r_us_u+x'}g^{z'}, \tag{13}$$

$$v'=g_2^{-r_ms_m+y'}g^{w'}, \tag{14}$$

$$u_i=g_2^{x_i}g^{z_i},\ 1\le i\le n \tag{15}$$

$$v_i=g_2^{y_i}g^{w_i},\ 1\le i\le n \tag{16}$$

where $F(ID*)=x'+\sum_{i=1}^n x_iID_i^*-r_us_u, J(ID^*)=z'+\sum_{i=1}^n z_iID_i^*, K(h*)=y'+\sum_{i=1}^n y_ih_i^*-r_ms_m,$ and $L(h*)=w'+\sum_{i=1}^n w_ih_i^*$ which satisfy $u'\prod_{i=1}^n u_i^{ID_i^*}=g_2^{F(ID*)}g^{J(ID*)}$ and $v'\prod_{i=1}^n v_i^{h_i^*}=g_2^{K(h*)}g^{L(h*)}$. And then send them to the adversary A.

S2 Private key extraction query. When the adversary A submits a query that the developer /modifier with the identity $ID*$ extracts the private key, Algorithm 1 does the followings:

- If $F(ID*)=0 \bmod r_u$, outputs "failure";
- If $F(ID*)\neq 0 \bmod r_u$), randomly chooses an integer $r_{ID*}\in Z_q^*$, and calculates the private key $d_{ID^*}^{(1)}, d_{ID^*}^{(2)}$ as

$$d_{ID^*}^{(1)}=g_1^{-J(ID*)/F(ID*)}(g_2^{F(ID*)}g^{J(ID*)})^{r_{ID}^*} \tag{17}$$

$$d_{ID^*}^{(2)}=g_1^{-1/F(ID*)}g^{r_{ID}^*} \tag{18}$$

And then, sends the private key d_{ID*} of the developer with its identity $ID*$ to the adversary A.

S3 APK file Authorization (Signature) query. When the adversary A submits a request that the developer/modifier with its identity $ID*$ signs an APK file, Algorithm 1 acts as follows:

- If $F(ID*)=0 \bmod r_u$, calls the private key extraction query to construct the private key of the developer/modify with the identity $ID*$, and then signs the APK file with the constructed private key;
- If $F(ID*) \neq 0 \bmod r_u$ and $K(h*) \neq 0 \bmod r_m$, chooses $r', r'' \in Z_q^*$, and calculates the signature $\sigma = (\sigma_1, \sigma_2, \sigma_3)$ as:

$$\sigma_1 = u'(\prod_{i=1}^{n} u_i^{ID*})^{r'} g_1^{-L(h*)/K(h*)} (v' \prod_{i=1}^{n} v_i^{h_i^*})^{r''}, \tag{19}$$

$$\sigma_2 = g^{r'}, \tag{20}$$

$$\sigma_3 = g_1^{-1/K(ID*)} g^{r''}, \tag{21}$$

-If $F(ID*)=0 \bmod r_u$ and $K(h*)=0 \bmod r_m$, outputs "failure".

S4 Authorization (Signature) forgery of the APK file. If Algorithm 1 does not terminate in solving process, the adversary A successfully outputs a signature $\sigma' = \{\sigma_1', \sigma_2', \sigma_2'\}$ of APK file (its coding is $h' = \{h_1', h_2', ..., h_n'\}$) signed by the developer/modify with the identity $ID*$, which can successfully forge an effective authorization (signature) of the APK file.

S5 Algorithm 1 solves an instance of CDH problem. If $F(ID*) \neq 0 \bmod q$ and $K(h') \neq 0 \bmod q$, Algorithm B terminates. Otherwise, if $F(ID*)=0 \bmod q$ and $K(h')=0 \bmod q$, a solution of an instance of CDH problem solved by Algorithm 1 is given by

$$g^{ab} = \frac{\sigma_1'}{(\sigma_2')^{J(ID')} (\sigma_3')^{J(h')}} \tag{22}$$

Note that the constructed private keys $d_{ID^*}^{(1)}, d_{ID^*}^{(2)}$ in Eq. (5) are valid because

$$\begin{aligned} d_{ID^*}^{(1)} &= g_1^{-J(ID*)/F(ID*)} (g_2^{F(ID*)} g^{J(ID*)})^{r_{ID}^*} \\ &= g_2^a (g_2^{F(ID*)} g^{J(ID*)})^{-a/F(ID*)} (g_2^{F(ID*)} g^{J(ID*)})^{r_{ID}^*} \\ &= g_2^a (g_2^{F(ID*)} g^{J(ID*)})^{r_{ID}^* - a/F(ID*)} \\ &= g_2^a (u' \prod_{i=1}^{n} u_i^{ID*})^{r_{ID}^*} \end{aligned} \tag{23}$$

and

$$\begin{aligned} d_{ID^*}^{(2)} &= g_1^{-1/F(ID^*)} g^{r_{ID}^*} \\ &= g_1^{-a/F(ID^*)} g^{r_{ID}^*} \\ &= g_1^{r_{ID}^* - a/F(ID^*)} \\ &= g_1^{\overline{r}_{ID^*}} \end{aligned} \tag{24}$$

To complete the proof, we need the following lemma:

Lemma 1. In order to ensure that there is at least ε' success probability to solve an instance of CDH problem, Algorithm B needs to satisfy the following three facts at the same time:

E1 When the adversary A requests the private key extraction, Algorithm 1 does not terminates, namely, $F(ID^*) \neq 0 \bmod r_u$;

E2 The adversary A can generate authorization (signature) of the APK file generated by the developer/modifier with the identity ID', which requires $K(h^*) \neq 0 \bmod r_m$;

E3 If the adversary A successfully forges the authorization (signature) of an APK file, which requires that they meet at the same time $F(ID')=0 \bmod q$ and $K(h')=0 \bmod q$, where $1 \leq i \leq n$;

Proof of Lemma 1. Let U_i be the event of $F(ID^*) \neq 0 \bmod r_u$, U^* be the event of $F(ID^*)=0 \bmod q$, and V_j be the event of $F(ID^*) \neq 0 \bmod q$ and $K(h')=0 \bmod q$. The success probability of Algorithm 1 is given by

$$\begin{aligned} \Pr[B_{succ}] &\geq \Pr[\cap_{i=1}^{q_e+q_s} U_i \wedge U^* \cap_{j=1}^{q_s} V_j \wedge V^*] \\ &\geq \Pr[\cap_{i=1}^{q_e+q_s} U_i \wedge U^*] \Pr[\cap_{j=1}^{q_s} V_j \wedge V^*] \end{aligned} \tag{25}$$

$$\geq \Pr[\cap_{i=1}^{q_e+q_s} U_i] \Pr[U^*] \Pr[\cap_{j=1}^{q_s} V_j] \Pr[V^*] \tag{26}$$

where Eqs. (25) and (26) are obtained from the independence of $U_i, U^*, V_j, V^*]$. Here, $\Pr[U^*]$ and $\Pr[\cap_{i=1}^{q_e+q_s} U_i]$ are calculated as follows:

$$\begin{aligned} \Pr[U^*] &= \Pr[F(ID') = 0 \bmod q \wedge F(ID') = 0 \bmod r_u] \\ &= \Pr[F(ID') = 0 \bmod r_u] \Pr[F(ID') = 0 \bmod q \mid F(ID') = 0 \bmod r_u] \\ &= \frac{1}{r_u} \times \frac{1}{n+1} \end{aligned} \tag{27}$$

Since

$$\begin{aligned} \Pr[\cap_{i=1}^{q_e+q_s} U_i] &= 1 - \Pr[\cup_{i=1}^{q_e+q_s} \neg U_i] \\ &\geq 1 - \frac{q_e + q_s}{n} \end{aligned}$$

So, we have

$$\Pr[\cap_{i=1}^{q_e+q_s} U_i \wedge U^*] \geq \frac{1}{r_u(n+1)} \times (1-\frac{q_e+q_s}{n}) \geq \frac{1}{4(q_e+q_s)(n+1)} \tag{28}$$

$$\Pr[\cap_{i=1}^{q_e+q_s} V_i \wedge V^*] \geq \frac{1}{4q_s(n+1)} \tag{29}$$

$$\Pr[B_{succ}] \geq \frac{1}{16q_s(q_s+q_e)(n+1)^2} \tag{30}$$

Hence, ε' is given $\varepsilon'=\varepsilon/(16q_s(q_e+q_s)(n+1)^2)$.

In the redevelopment and modification of APK, modifier can only modify the designated deposition or files of APK, as well as signs the modified APK files with the sanitizable signature. He cannot modify the unauthorized deposition or files of APK. Specifically, if adversary A can modify the unauthorized deposition or files of APK, namely those files encoded with base64, they have to meet time $K''=\{i \in K \mid h_i \neq h_i'\}$, and generate a sanitizable signature. It means that the propose APK-SAN authorization mechanism is not unforgettable secure. So, we obtain that:

Theorem 2. Assume there is a polynomial bounded adversary A that is able to break the immutability of APK-SAN authorization with an advantage ε' with time t, and q_e' private key extraction queries and q_s' signature queries. Then there exists an Algorithm 2 that can generate a valid signature with time t'' and advantage ε''.

Proof. The immutability of the authorized deposition of APK file means that: If advantage A deliberately or illegally modifies those files encoded with base64 and meets time $K''=\{i \in K \mid h_i \neq h_i'\}$ according to the signature verification we can discover the illegal modifications, and notice modified APK file is invalid. If adversary A is able to break the immutability of the authorized deposition of APK file, it means that Algorithm 2 is able to generate a valid signature. Algorithm 2 is defined as follows:

Algorithm 2

S1 Setup. **Algorithm 2** interacts with the challenger C as follows:

- **Algorithm 2** submits a request to the challenger C, and C returns system parameters (G_1, G_2, e, q, g, g_1, g_2, u', v', **u**, **v**);
- For $i \in K'$, **Algorithm 2** chooses $t_i \in Z_q^*$, and lets $u_i = g^{t_i}$ be a system parameter;
- For $i \in K'$, **Algorithm 2** chooses $s_i \in Z_q^*$, and lets $v_i = g^{s_i}$ be a system parameter.

S2 Private key extraction query. When A issues a private key of the modifier with the identity extraction query, Algorithm 2 acts as follows:

- Issues a private key query of the modifier with the identity *ID*', and gets the private key $(d_{ID_{i-1}}^{(1)}, d_{ID_{i-1}}^{(2)})$

– Let $\overline{d}_{ID_{i-1}}^{(1)}=d_{ID_{i-1}}^{(1)}(d_{ID_{i-1}}^{(2)})^{\prod_{i\in K'} ID_i t_i}$ and $\overline{d}_{ID_{i-1}}^{(2)}=d_{ID_{i-1}}^{(2)}$, and sends $(\overline{d}_{ID_{i-1}}^{(1)},\overline{d}_{ID_{i-1}}^{(2)})$ to the adversary A.

S3 Authorization (Signature) query. For the APK developed by the developer with the identity $ID*$, when the adversary A issues a redevelopment query encoded with $h'=\{h_1',h_2',...,h_n'\}$, Algorithm 2 acts as follows:

– Calculates

– $h_i^*=h_i' \ , i\in K''$, (31)

$$h_i^*=Base64(Hash(d*\,||\,H_i^*\,|\,ID*)), i\in K', \tag{32}$$

and represents as $h'=\{h_1^*,h_2^*,...,h_n^*\}$.

– Issues the redevelopment of APK and gets $\sigma_1,\sigma_2,\sigma_3$;

– Lets $\sigma_1^*=\sigma_1\prod_{i\in K''}\sigma_2^{ID_i^* t_i}\prod_{i\in K''}\sigma_3^{h_i^* s_i}, \sigma_2^*=\sigma_2, \sigma_3^*=\sigma_3$;

– Sends $\{\sigma_1^*,\sigma_2^*,\sigma_3^*\}\cup\{\sigma_3^{h_i^* s_i}, i\in K''\}$ to adversary A.

S4 Modification and Forgery. Assume that A is able to output a valid modification of the developer with the identity $\hat{ID}$ and APK file encoded as $\hat{h}=\{\hat{h}_1,\hat{h}_2,\ldots,\hat{h}_n\}$. Namely, the adversary A generates a new APK file and its signature $\{\hat{\sigma}_1,\hat{\sigma}_2,\hat{\sigma}_3\}$.

– A sends the modified signature $\{\hat{\sigma}_1,\hat{\sigma}_2,\hat{\sigma}_3\}$ of APK file encoded as $\hat{h}=\{\hat{h}_1,\hat{h}_2,\ldots,\hat{h}_n\}$ to **Algorithm 2**;

– Algorithm 2 constructs APK file developed by the developer with the identity $I\overline{D}$, and encoded as $\overline{h}=\{\overline{h}_1,\overline{h}_2,...,\overline{h}_n\}$. Namely he sets $\{ID_i\}$ and $\{h_i\}$, where $\overline{h}_i=\hat{h}_i, i\in K'$. Then outputs $h_i, i\in K'$. **Algorithm 2** constructs as follows

$$\overline{\sigma}_1=\frac{\hat{\sigma}_1}{\sigma_1\prod_{i\in K'}\hat{\sigma}_2^{\hat{ID}_i^*\hat{t}_i}\prod_{i\in K''}\sigma_3^{\hat{h}_i^*\hat{s}_i}}, \tag{33}$$

$$\overline{\sigma}_2=\hat{\sigma}_2, \tag{34}$$

$$\overline{\sigma}_3=\hat{\sigma}_3, \tag{35}$$

It is to verify that $\{\overline{\sigma}_1,\overline{\sigma}_2,\overline{\sigma}_3\}$ is APK file encoded as $\overline{h}_i=\hat{h}_i, i\in K''$, the signature signed by AS_i^* with the identity *ID*. According to definition 1, Algorithm 2 is able to solve an instance of CDH problem.

Obviously, the premise that Algorithm 2 successfully forges a signature of the APK file is that A is able to alter the immutability of the APK-SAN authorization. Therefore, Algorithm 2 has the success probability $\varepsilon''\geq\varepsilon$.

Meanwhile, the time t'' of Algorithm 2 is summation of A's running time t and the time which is used to respond to q_e'' private key extraction queries and q_s'' signature queries. Each private key extraction query requires Algorithm 2 to perform n exponentiation

operations in G_1. Each signature query requires Algorithm 2 to perform $2n$ exponentiation operations in G_1. We assume that the exponentiation operation in G_1 takes time t_e, Hence, the total time is $t'' = t + (nq'_e + 2nq'_s)t_e$.

5 Performances of the proposed scheme

In this section, we discuss the efficiency and security of the proposed scheme. For the cost of computation and storage, we only consider the following items: Signature overhead, verification overhead and increased size of the signature. For the computation complexity and memory space, we consider the computation costs of cryptographic operations involved in signature and verification schemes. For the space, we measure the memory cost, focus on the storage cost of key including the sizes of public key and private key.

5.1 The costs of computation and storage

Consider the scenario: After the developer accomplishes APK, a modifier applies for modifying the APK. The developer authorizes the modifier to improve or increase some functions in APK's specified location. According to the requirements of Android signature, the modifier can carry out two operations as following:

(1) After completing the modification, the modifier sends the modified APK to the developer. The developer signs and releases it;

(2) The modifier directly signs and releases the modified APK.

Table 1: The comparing of the overhead costs. SO denotes the signature overhead. VO denotes the verification overhead. IS denotes the increased size which means the size of extra signature file. R_d denotes the decryption overhead of one time RSA algorithm. R_e denotes the encryption overhead of one time RSA algorithm. T denotes the transmission overhead that an applicant returns to the designer. m denotes the number of APK files to modify. E_p denotes the exponentiation operation overhead. E denotes the logarithm operation overhead

Scheme	SO	VO	IS
Scheme 1 [Neidhardt (2011)]	$nR_d + T$	nR_e	N
APK-SAN scheme	$2mE_p$	$2mE_p+2E$	$3\|tt_1\|$

The operations in Scheme 1 [Neidhardt (2011)] belong to the first operation type, which are used to protect the integrity of APK file based on RSA algorithm. The operations in APK-SAN scheme belong to the second operation type. In this paper, we attempt to provide a scheme that the modifier is able to directly sign and release the modified APK. Meanwhile, one can verify the copyrights of APK. We assume that the Android signature is based on RSA algorithm, known as Scheme 1 [Neidhardt (2011)]. Their computation and storage costs are shown in Tab. 1. Here, each APK contains n files. P is a large prime integer used in this paper. Obviously, $m<n$.

From Tab. 1, the signature overhead of Scheme 1 [Neidhardt (2011)] is nR_d+T , and its verification overhead is nR_e. T denotes the time to finish the sending and receiving

processes. The transmission overhead is much higher than these of RSA encryption and decryption. The overhead of APK-SAN scheme comes from the pairing operations that depend on the number of the modified APK. If the 1024bits-RSA is adopted in Scheme 1 [Neidhardt (2011)], it costs 50 ms to finish a signature and 2.5 ms for verifying. Nevertheless, it only costs 43 ms to carry out the pairing operation in the finite field F_{97}. Obviously, APK-SAN scheme has less time overhead than that in Scheme 1. In the worst case, i.e. $m = 5$, the time overhead of APK-SAN scheme approximately equals to 430 ms in the signature phase and verification phase, respectively. These time costs are durable for smart mobile phones. On the other hand, a new signature file is generated because one needs sign APK files in Scheme 1 [Neidhardt (2011)]. The size of the signature is $|N|$, where $|N|$ is the digit of the modular number N. In APK-SAN scheme, the size of the signature is 3 $|G_1|$, where $|G_1|$ is the order of the finite group G_1. At the same time, Scheme 1 requires the certificate that increases the storage overhead. APK-SAN scheme is based on the user's identity without user's certificate.

5.2 Security

A fine-grained authorization means that the modifier can only modify the specified APK. Authentication means a scheme is able to verify the copyrights of the designer and the validity of the designer's and modifier's signatures. The over-privilege detection means that whether one scheme provides the over-privilege detection function or not. The copyrights protection means that whether one scheme has the ability to protect the copyrights after completing the modification.

As shown in Tab. 2, Scheme 1 [Neidhardt (2011)] only provides authentication for the developer, and needs the certificate which increases the storage overhead. Scheme 1 cannot provide a fine-grained authorization and Copyrights protection. The proposed APK-SAN scheme can provide all of these securities. In APK-SAN scheme, the developer authorizes a modifier, and specifies APK's files and their sub files that are allowed to be modified. Accordingly, it can restrict modifier's modification permissions, which prevent malicious codes from embedding in APK, and protect Copyrights of developer. If modifier violates the authorization regulations to modify the unauthorized APK's files, the modified APK's signature is invalid. Therefore, we can fulfill the over-privilege detection.

Table 2: The comparing of the security. FA denotes fine-grained authorization. OPD denotes the over-privilege detection. CP denotes the copyrights protection after modification

Scheme	FA	Authentication	OPD	CP
Scheme 1 [Neidhardt (2011)]	No	Yes	No	No
APK-SAN scheme	Yes	Yes	Yes	Yes

6 Conclusion

In this paper, we design a new APK signature APK-SAN with the unique properties of the sanitizable signature. New scheme allows developers of Android applications to authorize specified modifiers to modify designated source codes of APK file, and sign the modified APK file. The signature of new APK file is still valid. APK-SAN signature is designed to

ensure the integrity, authenticity and non-repudiation of APK files, while can reduce the communication time and computational overhead of developers. It also can maintain the legitimate rights and interests of developers and modifiers. Therefore, the proposed mechanism is able to solve the difficult problem that verifies the validity of the original designer's signature of APK and copyrights of APK when the authorization has completed.

Acknowledgement: This work was supported by the National Natural Science Foundation of China (No. 61662004, 61772437, 61702427), National Natural Science Foundation of Guangxi (No. 2016GXNSFAA380215), Sichuan Youth Science and Technique Foundation (No. 2017JQ0048), and EU ICT COST CryptoAction (No. IC1306).

References

Ateniese, G.; Chou, D. H.; DeMedeiros, B.; Tsudik, G. (2005): Sanitizable signatures. *European Symposium on Research in Computer Security*, pp. 159-177.

Barrera, D.; Kayacik, H. G.; van Oorschot, P. C.; Somayaji, A. (2010): A methodology for empirical analysis of permission-based security models and its application to Android. *Proceedings of the 17th ACM conference on Computer and Communications Security*, pp. 73-84.

Blasing, T.; Batyuk, L.; Schmidt, A. D.; Camtepe, S. A.; Albayrak, S. (2010): An Android application sandbox system for suspicious software detection. *5th International Conference on Malicious and Unwanted Software*, pp. 55-62.

Brzuska, C.; Fischlin, M.; Freudenreich, T.; Lehmann, A.; Page, M. et al. (2009): Security of sanitizable signatures revisited. *International Workshop on Public Key Cryptography PKC 2009*, pp. 317-336.

Chen, Y.; Ying, L.; Jiao, S.; Su, P.; Feng, D. (2014): Research on user privacy leakage of mobile social application. *Chinese Journal of Computers*, vol. 37, no. 1, pp. 87-100.

Choi, J.; Sung, W.; Choi, C.; Kim, P. (2015): Personal information leakage detection method using the inference-based access control model on the Android platform. *Pervasive and Mobile Computing*, vol. 24, pp. 138-149.

Dimitriadis, A.; Efraimidis, P. S.; Katos, V. (2016): Malevolent app pairs: An Android permission overpassing scheme. *ACM International Conference on Computing Frontiers*, pp. 431-436.

Egele, M.; Brumley, D.; Fratantonio, Y.; Kruegel, C. (2013): An empirical study of cryptographic misuse in android applications. *Proceedings of ACM SIGSAC Conference on Computer & Communications Security*, pp. 73-84.

Enck, W.; Damien, O.; McDaniel, P. D.; Chaudhuri, S. (2011): A study of Android application security. *USENIX Security Symposium*, vol. 2, pp. 2.

Felt, A. P.; Wang, H. J.; Moshchuk, A.; Hanna, S.; Chin, E. (2011): Permission re-delegation: attacks and defenses. *Proceedings of the 20th USENIX Security Symposium*, pp. 1-16.

Jeon, J.; Micinski, K. K.; Vaughan, J. A.; Fogel, A.; Reddy, N. et al. (2012): Android

and hide: Fine-grained permissions in Android applications. *ACM CCS Workshop on Security and Privacy in Smartphones and Mobile Device*, pp. 3-14.

Karim, M. Y.; Kagdi, H.; Penta, M. D. (2016): Mining Android Apps to recommend permissions. *IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering*, pp. 427-437.

Kim, J.; Yoon, Y.; Yi, K.; Shin, J.; Center, S. (2012): SCANDAL: Static analyzer for detecting privacy leaks in Android applications. *Mobile Security Technologies*.

Ma, S.; Li, L. T.; Deng, R. H. (2016): CDRep: Automatic repair of cryptographic misuses in Android applications. *Proceedings of 11th ACM Asia Conference on Computer and Communications Security*, pp. 711-722.

Mahmood, R.; Esfahani, N.; Kacem, T.; Mirzaei, N.; Malek, S. et al. (2012): A whitebox approach for automated security testing of Android applications on the cloud. *IEEE 7th International Workshop on Automation of Software Test*, pp. 22-28.

Neidhardt, E. (2011): Asymmetric cryptography for mobile devices. *Service Centric Networking*, pp. 1-12.

Park, J. K.; Choi, S. Y. (2015): Studying security weaknesses of Android system. *International Journal of Security and Its Applications*, vol. 9, no. 3, pp. 7-12.

Pearce, P.; Nunez, G.; Wagner, D. (2012): Addroid: Privilege separation for applications and advertisers in Android. *Proceedings of the 7th ACM Asia Conference on Computer and Communications Security*, pp. 71-72.

Qin, S. (2016): Research progress of Android security. *Journal of Software*, vol. 27, no. 1, pp. 45-71.

Qu, Z.; Keeney, J.; Robitzsch, S.; Zaman, F.; Wang, X. (2016): Multilevel pattern mining architecture for automatic network monitoring in heterogeneous wireless communication networks. *China Communications*, vol. 13, no. 7, pp. 108-116.

Shao, S.; Dong, G.; Guo, T.; Yang, T.; Shi, J. (2014): Modelling analysis and auto-detection of cryptographic misuse in android applications. *IEEE 12th International Conference on Dependable, Autonomic and Secure Computing*, pp. 75-80.

Silva, P.; Amcrim, V. J. P.; Ribeiro, F. N.; Muzetti, I. (2015): Privacy Mod: Controlling and monitoring abuse of privacy-related data by Android applications. *Brazilian Symposium on Computing Systems Engineering*, pp. 42-47.

Toorani, M.; Beheshti, A. A. (2008): LPKI-a lightweight public key infrastructure for the mobile environments. *IEEE Singapore International Conference on 11th Communication Systems*, pp. 162-166.

Waters, B. (2005): Efficient identity-based encryption without random oracles. *Advances in Cryptology*, pp. 114-127.