Android App Copy Protection Mechanism with Semi-trusted Loader

Kuo-Yu Tsai

Department of Applied Mathematics, Chinese Culture University, Taiwan
nicklastsai@gmail.com

Abstract

In this paper, we propose an Android App copy protection mechanism with a semi-trusted loader. In the proposed mechanism, an Android App is composed of an APK (application package file) file and a JAR (Java archive) file. As a mobile device user wants to purchase an Android App from the market, he/she has to download the APK file from the market and installs the APK in his/her device. At the first execution time, the embedded semi-trusted loader will download the encrypted JAR file from the market, and the corresponding decryption key for the encrypted JAR file. Then, the semi-trusted loader decrypts the JAR file by using the decryption key, and further, executes the loading for all functionalities. After the loading, the semi-trusted loader will delete the decryption key and the JAR file, and then store the encrypted JAR file in the mobile device. After that, the semi-trusted loader only download the decryption key from the market as the mobile user wants to execute all functionalities of the App. We adopt the signature scheme to protect the embedded semi-trusted loader in our proposed mechanism. As the semi-trusted loader attempts to download the decryption key, the market verifies if the semi-trusted loader is modified by verifying the signature.

Keywords: Android App, Copy protection, Dynamic loading, Semi-trusted loader

1 Introduction

According to Gartner's report [6], the downloaded times of mobile Apps will be more than 268 billion, and the App market scale will be more than $77 billion by 2017. More and more mobile Apps are designed, created and deployed to mobile devices, such as smart phones and tablets. However, there are various security threats and potential attacks on the Android vulnerabilities, and they have plagued mobile App developers and mobile App markets [1, 9]. Recently, Google announced two security mechanisms to solve various piracy problems for Android Apps, including the file permissions control mechanism and the license verification library mechanism. In the file permissions control mechanism, an App's APK (application package file) file will be stored in the /data/app folder after the Android App is installed in a mobile user's device. Only the installed App can access the folder [3, 7]. In the license verification library mechanism, the App will submit a query to the Google license server for obtaining the license status as a mobile user wants to execute his/her Android App on his/her mobile device. According to the license status, it allows or disallows the mobile user to execute the App on his/her mobile device [3, 7]. However, the above mechanisms cannot resist against the root attack on Android. Suppose that a malicious mobile user attempts to root his/her device for obtaining the folder permission. After obtaining the folder permission, the adversary can disassemble the APK and modify the APK file to disable the licensing service. Other related anti-piracy works [2, 8, 11-17, 21] adopt various techniques to prevent the Android Apps from unauthorized duplication, such as asymmetric and symmetric encryption, steganographic techniques (fingerprinting and watermarking), dynamic loading etc.

Kim [14] presented a copy protection system for Android Apps based the on public key infrastructure. To protect the Android Apps from illegal duplication, all Apps are encrypted by using different symmetric keys in Kim's proposed copy protection system. As a mobile user purchases an Android App, he/she will obtain an App encrypted by using the advanced encryption standard (AES for short) [5], and the corresponding AES decryption key is encrypted by using the mobile user's RSA public key [19]. Only the legitimate mobile user can recover the AES decryption key with his/her private key. Adopting the same concept, Moon *et al.* [17] also proposed a copyright protection system with the asymmetric encryption and symmetric encryption techniques for the Android platform. Kim [14] and Moon *et al.* [17] claimed that their proposed mechanisms can prevent the Apps form be misused by illegal mobile users, respectively. However, the APK file is stored in the /data/app folder after the App is installed in the mobile user's device. A malicious mobile user may illegally replicate the APK

*Corresponding Author: Kuo-Yu Tsai; E-mail: nicklastsai@gmail.com
DOI: 10.3966/160792642018011901006

file and distribute the file to other unauthorized mobile users.

For detecting illegal duplication of Android Apps, Ji and Kim [12] presented a mobile inspector by using fingerprinting techniques, in which an mobile inspector and an inspector helper are used to detect illegal Apps and pursue malicious distributors. The mobile inspector is used to inspect if an illegal App is installed on the mobile device. the inspector helper is responsible for extracting and delivering the necessary information for the mobile inspector. As a mobile user purchases an Android App, the Android market inserts the mobile user's information into the APK file by using the fingerprinting library. The mobile inspector determines if the App is illegal according to the same fingerprinting library as the Android market. In the same year, Jang *et al.* [8] proposed steganography-based software watermarking scheme to protect Android Apps, in which a watermark is divided into small bit strings of the same size and each bit string is encoded into multiple bit strings, using the Chinese Remainder Theorem [4]. The encoded bit strings are embedded by reordering the sequence of instructions in the basic blocks in Dalvik executable files. However, Ji and Kim's design [12] and Jang *et al.*'s scheme [8] cannot resist transformation attacks.

Jeong *et al.* [11] proposed an anti-privacy mechanism with class separation and dynamic loading in 2012. In Jeong *et al.*'s proposed mechanism, an Android App consists of an incomplete main App file (IMA for short) and a separate essential class file (SEC for short). As a mobile user purchases an Android App, he/she has to download the IMA file from the Android market and installs the IMA file on his/her mobile device. The Android market will send the encrypted SEC file to the mobile user when the mobile user first executes the IMA. The downloaded encrypted SEC file is decrypted and stored in the secure space, and then SEC file is dynamic loaded for execution. Based on dynamic loading, Jeong *et al.* [10] proposed an integrity check approach to prevent execution for unauthorized Apps, in which a MD5 hashing value [18] is used to ensure the integrity of the SEC file. However, both of Jeong *et al.*'s proposed mechanism [11] and Jeong *et al.*'s proposed approach [10] cannot resist against the root attack on the Android platform. A malicious mobile user may root his/her mobile device, and hence access the secure space for replicating the decrypted SEC file.

Inspired from Moon *et al.*'s mechanism [17] and Jeong *et al.*'s mechanism [11], Tsai *et al.* [20] proposed an Android App copy protection mechanism with the dynamic loading function. In Tsai *et al.*'s proposed Android App copy protection mechanism, a complete Android App also includes a SEC file and an IMA file. As a mobile user purchases an Android App from the Android market, the mobile user has to download and install the IMA file on his/her mobile device. When the mobile user wants to executes the IMA, the dynamic loading function will download the SEC file from the Android market and load the SEC file into the App's address space for enabling all functionalities. In addition, the dynamic loading function will delete the SEC file after the loading. However, Tsai *et al.*'s proposed mechanism cannot also resist against the root attack on the Android platform. Suppose that a malicious mobile user attempts to root his/her device to replicate the APK. He/she can disassemble the APK and modify the dynamic loading function of the APK file to disable the deletion capability. Hence, the malicious user can replicate the SEC file, and illegally distribute the SEC file. In addition, the mobile user has to download the encrypted SEC file as he/she wants to execute the IMA.

In this paper, we employ a semi-trusted loader which may misbehave on its own, but the misbehavior is detectable. In our proposed mechanism, we adopt the signature scheme to ensure the integrity of the embedded semi-trusted loader. At the first executing time, the embedded semi-trusted loader will download the encrypted SEC file from the market and the corresponding decryption key for the encrypted SEC file. Then, the semi-trusted loader decrypts the encrypted SEC file by using the decryption key and executes the loading for all functionalities. After the loading, the semi-trusted loader will delete the decryption key and the SEC file, and only store the encrypted SEC file in the mobile device. After that, the semi-trusted loader only downloads the decryption key from the market as the mobile user wants to execute the IMA.

The rest of this paper is organized as follows: In Section 2, we reviews and analyze Tsai *et al.*'s proposed Android App Copy Protection mechanism. Section 3 presents our proposed Android App copy protection mechanism to prevent mobile Apps from illegal duplication. Security analysis of our proposed mechanism and discussions are given in Section 4. Section 5 presents conclusions.

2 Tsai et al.'s Android App Copy Protection Mechanism

Table 1 lists the symbols used in the subsequent descriptions.

Tsai *et al.*'s [20] proposed Android App copy protection mechanism consists four phases: ***Registration Phase***, ***App Uploading Phase***, ***App Purchase Phase***, and ***App Execution Phase***. Detailed descriptions are as follows.

2.1 Registration Phase

In this phase, a mobile user MU_i has to prepare his/her identity MU_i, password pw_i, and international mobile equipment identity $IMEI_i$ and registers with the

Table 1. Symbols

Symbol	Interpretation
AM	App market
ASE	App security enhancer
AD	App developer
MU_i	Mobile user MU_i who purchases an Android App and executes the App
pwi	Mobile user MU_i's password
IMEIi	International mobile equipment identity of the mobile user MU_i's mobile device
UL_i	Mobile user MU_i's license
K_{AM}	Secret key shared between *AM* and MU_i
K_{ASE}	Secret key shared between *ASE* and MU_i
D_K/E_K	Symmetric decryption and encryption algorithms using the same key *K*
S/*V*	Signature generation and verification algorithms
DLF	Dynamic loading function
AID	App identity
APK	Application package file
T	Timestamp
SC	Separated class
H	One-way hashing function

App market *AM* via a secure channel. Detailed descriptions are as follows. (see Figure 1)

Step1: The MU_i sends his/her registration information $\{MU_i, pw_i, IMEI_i\}$ to the *AM*.

Step2: After receiving $\{MU_i, pw_i, IMEI_i\}$, the *AM* generates a license UL_i and two session keys $\{K_{AM}, K_{ASE}\}$ for the MU_i.

Setp3: The AM sends the secret information $\{UL_i, K_{AM}, K_{ASE}\}$ to the MU_i.

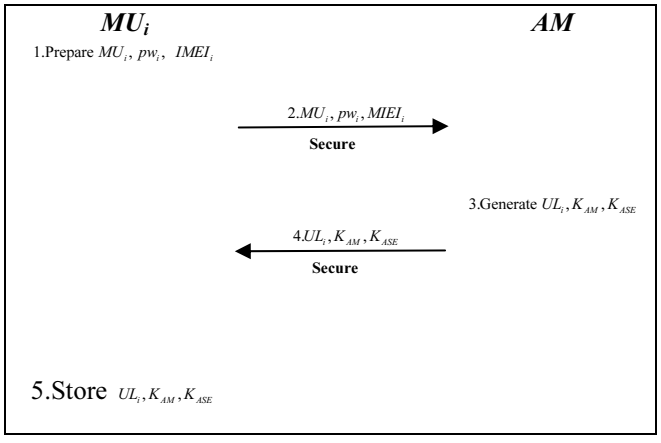

Figure 1. Registration phase

2.2 App Uploading Phase

An App developer *AD* develops a complete App which is composed of a separated class *SC* and an incomplete main App *APK*. The *APK* includes the dynamic loading function *DLF*, which is responsible for loading a *SC*. The *SC* includes the other additional functionalities of the App. The *AD* uploads the *APK* and the corresponding App identity *AID* into the *AM* and the *SC* into the *ASE*, respectively.

2.3 App Purchase Phase

As a mobile user MU_i wants to purchase an App with the App identity *AID*, the MU_i and the *AM* cooperate to perform the following steps. (see Figure 2)

Step1: MU_i encrypts *AID*, UL_i, pw_i, and $IMEI_i$ by using his/her secret key K_{AM} :

$$E_{K_{AM}}(UL_i || AID || pw_i || IMEI_i).$$

Step2: MU_i sends $\{E_{K_{AM}}(UL_i || AID || pw_i || IMEI_i), MU_i\}$ to the *AM*.

Step3: Upon receiving the purchase information, the *AM* performs the decryption operation to obtain *AID*, UL_i, pw_i, and $IMEI_i$:

$$D_{K_{AM}}(E_{K_{AM}}(UL_i || AID || pw_i || IMEI_i)).$$

Setp4: The *AM* verifies the decrypted UL_i, pw_i, and $IMEI_i$. If all messages are valid, the *AM* continues to perform Step 5; otherwise, he/she rejects the request and returns the failed information.

Step5: The *AM* generates a signature by using his/her private key:

$$Sig_{AM} = S_{AM}(H(APK || AID || UL_i || IMEI_i)).$$

Step6: The *AM* encrypts *APK* and Sig_{AM} by using the secret key K_{AM}:

$$E_{K_{AM}}(APK || Sig_{AM} || IMEI_i).$$

Step7: The *AM* sends $E_{K_{AM}}(APK || Sig_{AM} || IMEI_i)$ to the MU_i.

Step8: The MU_i decrypts the received data to obtain *APK*:

$$D_{K_{AM}}(E_{K_{AM}}(APK || Sig_{AM} || IMEI_i)).$$

Step9: The MU_i installs the *APK* in his/her mobile device.

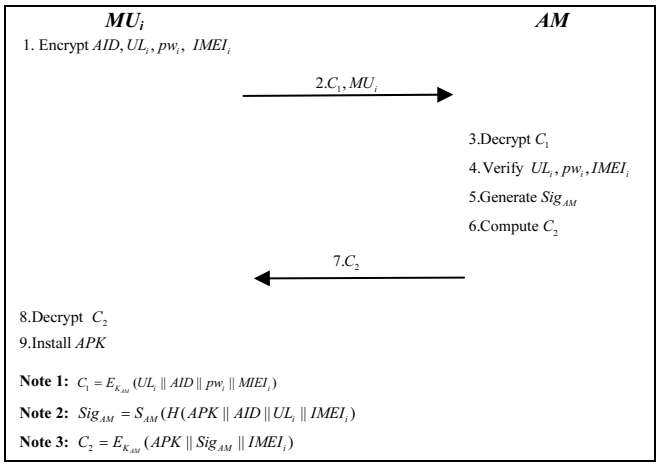

Figure 2. App purchase phase

2.4 App Execution Phase

When the MU_i wants to execute his/her purchased App, the *DLF* and the *ASE* cooperate to perform the following steps. (see Figure 3)

Step1: The *DLF* encrypts UL_i, T_i, Sig_{AM}, *AID*, and $IMEI_i$ as:

$E_{K_{ASE}}(UL_i|| T_i|| Sig_{AM}|| AID|| IMEI_i)$,

where T_i is the timestamp at the login mobile device.

Step2: The *DLF* sends the authentication infomration $\{E_{K_{ASE}}(UL_i|| T_i|| Sig_{AM}|| AID|| IMEI_i), UL_i\}$ to the *ASE*.

Step3: Upon receiving the information, the *ASE* performs the decryption operation to obtain UL_i, T_i, Sig_{AM}, *AID*, and $IMEI_i$:

$D_{K_{ASE}}(E_{K_{ASE}}(UL_i|| T_i|| Sig_{AM}|| AID|| IMEI_i))$.

Step4: The *ASE* performs the following authentication process.

Step4-1: Verify if T_i is valid, then continue to perform Step 4-2; Otherwise, reject the request.

Step4-2: Verify if UL_i and the corresponding $IMEI_i$ correct, then continue to perform the next step; Otherwise, reject the request.

Step4-3: Perform the signature verification operation:

$V_{AM}(S_{AM}(H(APK|| AID|| UL_i || IMEI_i)))$.

If all verification steps are correct, it means that the received execution request is new, and the legal user MU_i executes his/her App on the specified mobile device with the $IMEI_i$; Otherwise, the *ASE* rejects MU_i's execution request.

Step5: The *ASE* generates the signature Sig_{ASE}:

$Sig_{ASE}= S_{ASE}(H(UL_i|| T'_i|| AID|| SC|| IMEI_i))$, where T'_i is the transmission timestamp sent by the *ASE*, and the *SC* is a separated class.

Step6: The *ASE* encrypts T'_i, *SC*, and Sig_{ASE} as:

$E_{K_{ASE}}(T'_i|| SC|| Sig_{ASE})$

Step7: The *ASE* sends $E_{K_{ASE}}(T'_i|| SC|| Sig_{ASE})$ to the *DLF*.

Step8: After receiving the transmitted data, the *DLF* decrypts $E_{K_{ASE}}(T'_i|| SC|| Sig_{ASE})$ to obtain T'_i, *SC*, and Sig_{ASE}.

Step9: The *DLF* performs the following verification process.

Step9-1: Verify if T'_i is valid, then continue to perform the next step; Otherwise, reject to perform Step9-2.

Step9-2: Perform the signature verification operation:

$V_{ASE}(S_{ASE}(H(UL_i|| T'_i|| AID|| SC|| IMEI_i))$.

If all verification steps are correct, it means that the MU_i is a legal mobile user; otherwise, the *DLF* rejects to load the *SC*.

Step10:The *DLF* loads the *SC*, and then, the MU_i to execute all functionalities of the App.

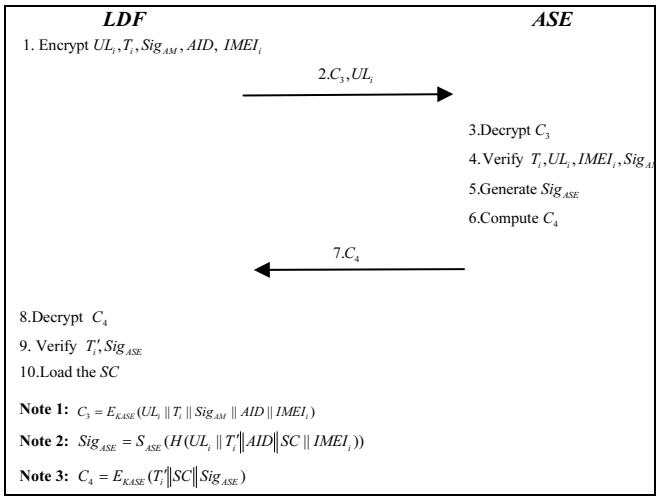

Figure 3. App execution phase

2.5 Discussions

To prevent the *SC* from illegal duplication, the *DLF* will delete the *SC* in the mobile device as the functionalities are loaded. However, a malicious mobile user may root his/her device and replicate the APK stored in the /data/app folder. Futher, the malicious mobile user can disassemble the APK and modify the dynamic loading function to disable the deletion capability. Hence, the *SC* will be stored in the mobile device, and the malicious user can replicate the SEC file, and illegally distribute it to other unauthorized mobile users. In addition, the *DLF* has to download the encrypted *SC* as the mobile user wants to execute the App.

3 Our Proposed Android App Copy Protection Mechanism

Table 2. Additional symbols

Symbol	Interpretation
STL	Semi-trusted loader
$Char_{STL}$	Characteristics for the *STL*

Most of the symbols used in our proposed mechanism are the same as ones used in Tsai *et al.*' proposed mechanism [20]. The dynamic loading function in Tsai *et al.*' proposed mechanism may be modified by a malicious mobile user. Hence, we employ a semi-trusted loader which may misbehave on its own, but the misbehavior is detectable. Our proposed consists five phases: ***Registration Phase***, ***Android App Uploading Phase***, ***App Purchase Phase***, ***App First Executing Phase***, and ***App Second Executing Phase***. The descriptions of ***Registration Phase*** are the same as ones in Tsai *et al.*' proposed mechanism. Detailed descriptions of other phases are follows.

3.1 Android App Uploading Phase

An App developer *AD* develops a complete App consisting of a separated class *SC* and an incomplete main App *APK*. The APK file includes the semi-trusted loader *STL* and the characteristics $Char_{STL}$ for the *STL*. The *STL* is responsible for loading a separated class *SC* and deleting *SC* and secure keys. The *SC* includes the other additional functionalities of the App. The *AD* generates a signature $Sig_{AD}= S_{AD}(H(APK|| AID|| Char_{STL}))$ and uploads the *APK* and the corresponding App identity *AID* into the *AM* and the *SC* and Sig_{AD} into ASE, respectively.

3.2 App Purchase Phase

Most steps are the same in Tsai *et al.*' proposed mechanism [20], except Step 5 and Step 6, as follows. (see Figure 4)

Step5: The *AM* generates a signature by using his/her private key:
$Sig_{AM}= S_{AM}(H(APK|| AID|| Sig_{AD}|| UL_i || IMEI_i))$.

Step6: The *AM* encrypts *APK* and Sig_{AM} by using the secret key K_{AM}:
$E_{K_{AM}}(APK|| Sig_{AM}|| Sig_{AD}|| IMEI_i)$.

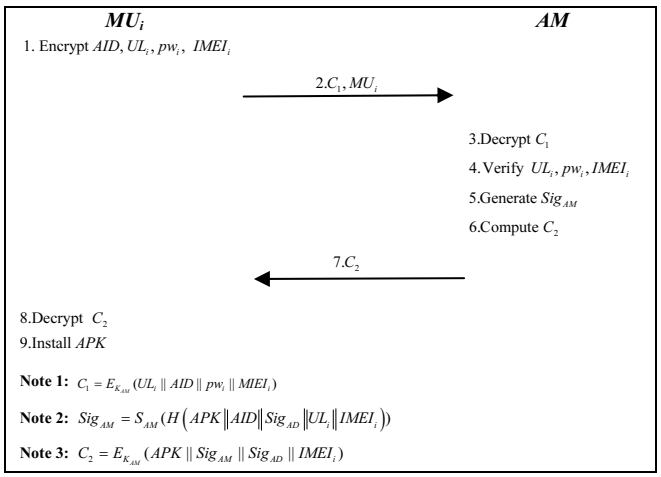

Figure 4. App purchase phase

3.3 App First Executing Phase

As the MU_i wants to execute all functionalities of his/her purchased App, the *STL* embedded in the App and the *ASE* will perform the authentication process. (see Figure 5)

Step1: The *STL* generates the ciphertext $E_{K_{ASE}}(UL_i|| T_i|| Sig_{AM}|| Sig_{AD}|| AID|| IMEI_i)$, where T_i is the timestamp for the transmission performed by the *STL*.

Step2: The *STL* sends the authentication information $\{E_{K_{ASE}}(UL_i|| T_i|| Sig_{AM}|| Sig_{AD}|| AID|| IMEI_i), UL_i\}$ to the ASE.

Step3: After receiving $\{E_{K_{ASE}}(UL_i|| T_i|| Sig_{AM}|| Sig_{AD}|| AID|| IMEI_i), UL_i\}$, the *ASE* decrypts the received messages to obtain UL_i, T_i, Sig_{AM}, Sig_{AD}, *AID*, and $IMEI_i$.

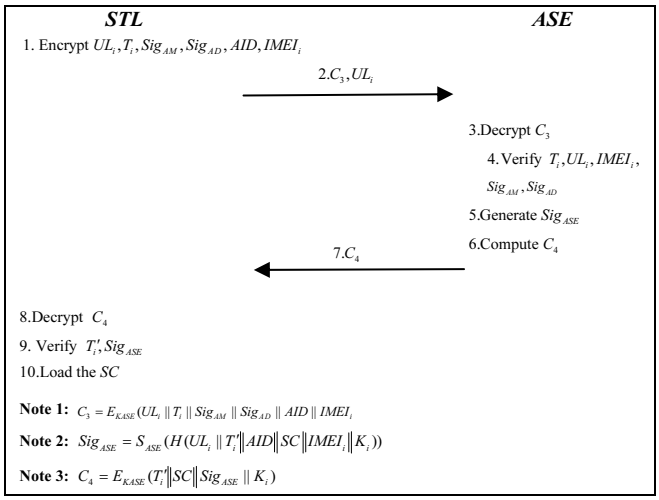

Figure 5. App first executing phase

Step4: The *ASE* performs the following steps to verify if the mobile user MU_i is legal and the *STL* is not modified by anyone.

Step4-1: Verify if T_i is valid, then continue to perform Step 4-2; Otherwise, reject the request.

Step4-2: Verify if UL_i and the corresponding $IMEI_i$ are correct, then continue to perform the next step; Otherwise, reject the request.

Step4-3: Perform the signature verification operation:
$V_{AM}(S_{AM}(H(APK|| AID|| Sig_{AD}|| UL_i || IMEI_i)))$.
If the signature verification is correct, it means that MU_i is a legal mobile user, and the *ASE* continues to perform Step 4-4; Otherwise, he/she rejects the execution request.

Step4-4: Retrieve the $Char_{STL}$ from the *STL*.

Step4-5: Perform the signature verification operation:
$V_{AD}(S_{AD}(H(APK|| AID|| Char_{STL})))$.
If the above signature verification is also correct, it ensures the integrity of the *STL*, and the *ASE* continues to perform Step 5; Otherwise, the *ASE* rejects the execution request.

Step5: The *ASE* generates the signature $Sig_{ASE}= S_{ASE}(H(UL_i|| T'_i|| AID|| SC|| IMEI_i|| K_i))$, where T'_i is the transmission timestamp sent by the *ASE*, and K_i is the short-term secret key for encrypting the *SC*.

Step6: The *ASE* preforms the encryption operation $E_{K_{ASE}}(T'_i|| SC|| Sig_{ASE}|| K_i)$.

Step7: The *ASE* sends $E_{K_{ASE}}(T'_i|| SC|| Sig_{ASE}|| K_i)$ to the *STL*.

Step8: Upon receiving $E_{K_{ASE}}(T'_i|| SC|| Sig_{ASE}|| K_i)$, the *STL* decrypts $E_{K_{ASE}}(T'_i|| SC|| Sig_{ASE}|| K_i)$ to obtain T'_i, *SC*, Sig_{ASE}, and K_i.

Step9: The *STL* performs the following steps.

Step9-1: Verify if T'_i is valid, then continue to perform Step9-2; Otherwise, reject it.

Step9-2: Perform the signature verification operation $V_{ASE}(S_{ASE}(H(UL_i \| T'_i \| AID \| SC \| IMEI_i \| K_i)))$.

Step10: The *STL* loads the *SC* to allow the MU_i to execute all functionalities of the App.

Step11: The *STL* deletes the *SC* and K_i in the mobile device and store $E_{K_i}(T'_i \| SC \| Sig_{ASE})$ in the secure space.

3.4 App Second Executing Phase

After the first time for executing the App, the *STL* embedded in the App performs the authentication process as the MU_i wants to execute all functionalities of his/her purchased App. (see Figure 6)

Step1: The *STL* generates the authentication information $E_{K_{ASE}}(UL_i \| T''_i \| Sig_{AM} \| AID \| IMEI_i)$, where T''_i is a timestamp for the transmission performed by the *STL*.

Step2: The *STL* sends $\{E_{K_{ASE}}(UL_i \| T''_i \| Sig_{AM} \| AID \| IMEI_i), UL_i\}$ to the *ASE*

Step3: Upon receiving the information, the *ASE* decrypts $E_{K_{ASE}}(UL_i \| T''_i \| Sig_{AM} \| AID \| IMEI_i)$ and obtains UL_{MU}, T_{MU}, Sig_{AM}, *AID*, and *IMEI*.

Step4: The *ASE* performs the same steps described in *App First Executing Phase* to ensure the legality of the MU_i and integrity of the *STL*.

Step5: The *ASE* generates the encryption $E_{K_{ASE}}(T'''_i \| K_i)$, where T'''_i is a timestamp for the transmission performed by the *ASE*; Otherwise, the *ASE* rejects MU_i's App execution request.

Step6: The *ASE* sends $E_{K_{ASE}}(T'''_i \| K_i)$ to the *STL*.

Step7: After receiving $E_{K_{ASE}}(T'''_i \| K_i)$, the *STL* decrypts $E_{K_{ASE}}(T'''_i \| K_i)$ to obtain K_i.

Step8: The *STL* verifies if T'''_i is valid, then the *STL* continues to perform Step 9-2; Otherwise, reject it.

Step9: The *STL* uses K_i to decrypt $E_{K_i}(T_{ASE} \| SC \| Sig_{ASE})$.

Step10: The *STL* loads the *SC* to allow the MU_i to execute all functionalities of the App.

Step11: The *STL* deletes the *SC* and K_i in MU_i's mobile device.

4 Analysis and Discussions

4.1 Security Analysis

Our proposed mechanism achieves resistance to illegal copy, unforgeability, resistance to the man-in-the middle attack, and resistance to the replay attack. Detailed description are as follows.

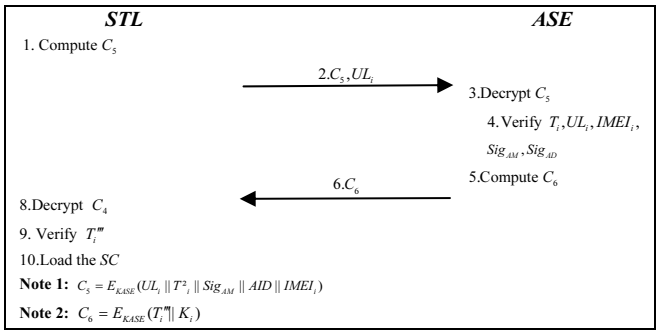

Figure 6. App Second Executing Phase

Resistance to illegal copy. In our experiment, the *STL* can dynamically load and delete the *SC* and decryption key K_i. Hence, a malicious user cannot illegally obtain the *SC* even though he/she has rooted his/her mobile device.

Unforgeability. Consider two scenarios: (1) An adversary may try to cheat the ASE and claim that he/she has purchase some App. However, the adversary cannot successfully forge $Sig_{AM} = S_{AM}(H(APK \| AID \| UL_{MU} \| IMEI))$ without the AM's private key to pass the verification process performed by the ASE. (2) An adversary may try to cheat the *STL* and claim that he/she has purchase some App for the specified mobile device. However, the adversary cannot successfully forge $S_{ASE}(H(UL_{MU} \| T_{ASE} \| AID \| SC \| IMEI))$ due to he/she does not have the ASE's private key.

Resistance to the man-in-the middle attack. Suppose that an adversary attempts to decrypt the data in the App purchase and the App execution phases. However, he/she cannot obtain the decrypted data without K_{AM} and K_{ASE}.

Resistance to the replay attack. Assume that an adversary attempts to intercept the transmitted data and replay them. However, he/she cannot success due to each transmitted data includes a timestamp.

4.2 Comparison

In comparison of Jeong *et al.*'s mechanism [11], Jeong *et al.*'s mechanism [10] and Tsai *et al.*'s mechanism [20], our proposed mechanism can detect the root attack on the Android platform by verifying the characteristics for the *STL*. Note that our proposed mechanism is slightly outperformed, but our proposed mechanism can achieve all functionalities listed in Table 3.

Table 3. Comparison of functionalities

	F_1	F_2	F_3
Our proposed mechanism	Yes	Yes	Yes
Tsai *et al.*'s mechanism [20]	No	Yes	Yes
Jeong *et al.*'s mechanism [11]	No	Yes	No
Jeong *et al.*'s mechanism [10]	No	Yes	Yes

Note: **F_1:** providing the root attack detection.
F_2: providing the dynamic loading
F_3: ensuring the integrity of the *SC*

4.2 Prototype Implementation

To prove the practicability of our proposed mechanism, we implement the prototype based on the Android platform. As the MU_i wants to execute his/her purchased App, he/she has to input his/her identity and password. (as shown in Figure 7(a))

(a) input MU_i and pw_i (b) verifcaion result: legal mobile user (c) verifcaion result: pirate user

Figure 7. The prototype implementation of our proposed mechanism

Then, the *STL* embedded in the App and the *ASE* will perform the authentication process. If the verification is successful, it means that the integrity of the *STL* and the legality of the MU_i are verified by the *ASE*. The *ASE* will send the information. (shown as in Figure 7(b))

5 Conclusion

In this paper, we propose an Android App copy protection mechanism with a semi-trusted loader. In our proposed mechanism, it is detectable as the semi-trusted loader is modified by a malicious user. In addition, we also prove the concept for our proposed scheme by implementing a prototype in the Android platform.

Our proposed mechanism is slightly outperformed, and our future work is to design App copy protection mechanism by using lightweight cryptographic techniques.

Acknowledgment

The authors gratefully acknowledge the support from Taiwan Information Security Center (TWISC) and National Science Council under the grants 103-2221-E-146-005 -MY2 and 103-2221-E-011-090-MY2 in Taiwan.

References

[1] H. S. Choi, Y. A. Au, C. Z. Liu, Is Digital Piracy an Enemy of the Mobile App Industry? An Empirical Study on Piracy of Mobile Apps, *Proceedings of the 20th Americas Conference on Information Systems (AMCIS 2014)*, Savannah, GA, 2014, pp. 1-9.

[2] S. Choi, J. Jang, E. Jae, Android Application's Copyright Protection Technology Based on Forensic Mark, *Proceedings of the 2012 ACM Research in Applied Computation Symposium (ACM RACS 2012)*, San Antonio, TX, 2012, pp. 338-339.

[3] C.-Y. Chuang, Y.-C. Wang, Y.-B. Lin, Digital Right Management and Software Protection on Android Phones, *Proceedings of the IEEE 71st Vehicular Technology Conference (VTC 2010-Spring)*, Taipei, Taiwan, 2010, pp. 1-5.

[4] C. Ding, D. Pei, A. Salomaa, *Chinese Remainder Theorem: Applications in Computing, Coding, Cryptography*, World Scientific Publishing, 1996.

[5] Federal Information Processing Standards Publications, Advanced Encryption Standard, *FIPS 197*, 2001.

[6] Gartner Inc., Gartner Says by 2017, Mobile Users Will Provide Personalized Data Streams to More Than 100 Apps and services Every Day, http://www.gartner.com/newsroom/id/2654115, January, 2014.

[7] Google Inc., *Application Licensing*, http://developer.android.com/google/play/licensing/index.html, 2014.

[8] J. Jang, H. Ji, J. Hong, J. Jung, D. Kim, S. K. Jung, Protecting Android Applications with Steganography Based Software Watermarking, *Proceedings of the 28th Annual ACM Symposium on Applied Computing (ACM SAC 2013)*, Coimbra, Portugal, 2013, pp. 1657-1658.

[9] J. Jang, S. Han, Y. Cho, U J. Choe, J. Hong, Survey of Security Threats and Contermeasures on Android Environment, *Journal of Security Engineering*, Vol. 11, No. 1, pp. 1-12, 2014.

[10] Y.-S. Jeong, J.-C. Moon, D. Kim, S.-J. Cho and M. Park, Preventing Execution of Unauthorized Applications Using Dynamic Loading and Integrity Check on Android Smartphones, *Information*, Vol. 16, No. 8(A), pp. 5857-5868, August, 2013.

[11] Y.-S. Jeong, Y.-U. Park, J.-C. Moon, S.-J. Cho, D. Kim, M. Park, An Anti-piracy Mechanism Based on Class Separation and Dynamic Loading for Android Applications, *Proceedings of the 2012 ACM Research in Applied Computation Symposium (ACM RACS 2012)*, San Antonio, TX, 2012, pp. 328-332.

[12] H. Ji, W. Kim, Design of a Mobile Inspector for Detecting Illegal Android Applications Using Fingerprinting, *Proceedings of the 2013 Research in Adaptive and Convergent Systems (RACS 2013)*, Montreal, Quebec Province, Canada, 2013, pp. 363-364.

[13] N. T. Kannengiesser, U. Baumgarten, S. Song, Secure Copy Protection for Mobile Apps, *Proceedings of the 12th International Symposium on Ambient Intelligence and Embedded Systems (AmiEs-2013)*, Berlin, Germany, 2013.

[14] S.-R. Kim, Copy Protection System for Android App Using Public Key Infrastructure, *Journal of Security Engineering*, Vol. 9, No 1, pp. 121-134, 2012.

[15] B. Kim, J. Jung, Impact of Multiple Watermarks for Protecting Copyright of Applications on Smart Mobile

Devices, *Proceedings of the 2013 Research in Adaptive and Convergent Systems (RACS 2013)*, Montreal, Quebec Province, Canada, 2013, pp. 359-360.

[16] S. Kim, E. Kim, J. Choi, A Method for Detecting Illegally Copied APK Files on the Network, *Proceedings of the 2012 ACM Research in Applied Computation Symposium (ACM RACS 2012)*, San Antonio, TX, 2012, pp. 253-256.

[17] Y. C. Moon, J. H. Noh, A. R. Kim, S.-R. Kim, Design of Copy Protection System for Android Platform, *Proceedings of International Conference on Information Technology, System and Management (ICITSM 2012)*, Dubai, United Arab Emirates, 2012.

[18] R. L. Rivest, *The MD5 Message-Digest Algorithm*, *Request for Comments 1321*, Network Working Group, April, 1992.

[19] R. L. Rivest, A. Shamir, L. M. Adleman, A Method for Obtaining Digital Signatures and Public-Key Cryptosystems, *Communications of the ACM*, Vol. 21, No. 2, pp. 120-126, February, 1978.

[20] K.-Y. Tsai, Y.-H. Chiu, T.-C. Wu, Android App Copy Protection Mechanism Based On Dynamic Loading, *Proceedings of the 18th IEEE International Symposium on Consumer Electronics (ISCE 2014)*, Jeju, Korea, 2014, pp. 1-3.

[21] D. Tse, Z. Li, Y. Tao, K. F. Wong, W. H. Choi, W. Liu, Two-factor Protection Scheme in Securing the Source Code of Android Applications, *Proceedings of the 2nd BCS International IT Conference*, Abu Dhabi, United Arab Emirates, 2014, pp. 1-7.

Biography

Kuo-Yu Tsai is an Assistant Professor at the Department of Applied Mathematics, Chinese Cluture University, Taiwan. He received his Ph.D. Degree in Information Management from National Taiwan University of Science and Technology, Taiwan, in 2009. From 2009 to 2012, he was a post-doctor at the Taiwan Information Security Center, National Taiwan University of Science and Technology, Taiwan. He joined as an assistant professor in the Department of Management Information Systems, Hwa Hsia University of Technology from 2012 to 2016. His recent research interests include IoT security, mobile commerce and security, healthcare application and security, and cryptography.