Hindawi Publishing Corporation
The Scientific World Journal
Volume 2014, Article ID 295789, 6 pages
http://dx.doi.org/10.1155/2014/295789

Research Article

Software Authority Transition through Multiple Distributors

Kyusunk Han[1] and Taeshik Shon[2]

[1] *University of Michigan, 500 S. State Street, Ann Arbor, MI 48109, USA*
[2] *Department of Information Computer Engineering, Ajou University, San 5, Woncheon-dong, Yeongtong-gu, Suwon 443-749, Republic of Korea*

Correspondence should be addressed to Taeshik Shon; tsshon@ajou.ac.kr

Received 4 June 2014; Accepted 2 July 2014; Published 20 July 2014

Academic Editor: Sang-Soo Yeo

The rapid growth in the use of smartphones and tablets has changed the software distribution ecosystem. The trend today is to purchase software through application stores rather than from traditional offline markets. Smartphone and tablet users can install applications easily by purchasing from the online store deployed in their device. Several systems, such as Android or PC-based OS units, allow users to install software from multiple sources. Such openness, however, can promote serious threats, including malware and illegal usage. In order to prevent such threats, several stores use online authentication techniques. These methods can, however, also present a problem whereby even licensed users cannot use their purchased application. In this paper, we discuss these issues and provide an authentication method that will make purchased applications available to the registered user at all times.

1. Introduction

In recent years, software distribution models have changed rapidly. Apple's *iOS Appstore* and *iTunes* made a significant change to the ecosystem of software and content distribution.

The convenience of these systems inspired other competitors and solutions. For mobile devices, for example, Google launched *Google Play* for their Android OS, and Amazon have developed their own *Amazon Appstore*. The success of these endeavors has influenced the PC-based OS software ecosystem. Microsoft has recently released *Windows Store* for Windows 8, and Apple has released *Mac Appstore*.

Whereas Apple's iOS only allows access to their built-in store, most distributors allow users other options. For example, the Android system allows access to Google's built-in store service as well as other mobile carriers' store services, even including those manually installed by the user. While users can purchase apps from the Windows and Mac appstores, they can also purchase them from other distributors or developers as well.

Although such market services provide significant convenience to users, they have introduced several issues. With the traditional software purchase environment, users could obtain product support regardless of where they purchased their applications. Users who purchase an application from a specific online appstore, however, cannot get support if they cancel or lose their connection to the store. Moreover, if an app requires an online authentication process to verify a valid license, the user will not even be able to launch the app.

In this paper, we discuss the software authorization issue and propose an extended "purchase authentication service" (PAS) model [1] that ensures users are authorized to access applications even if they change their status. Our extended PAS avoids the use of an independent system, which can cause overheads. We demonstrate two scenarios: (1) users are using a roaming service and (2) users permanently change their contact details.

We present an overview of the online application store model in Section 2. In Section 3, we discuss problems with application authorization. We then propose, in Section 4, an authentication protocol that allows a user to obtain authorization from multiple vendors. We then analyze the security of the model in Section 5 and conclude this paper in Section 6.

2. Online Application Store

Commercial consumer software was traditionally distributed as a package through offline markets. Users purchased an

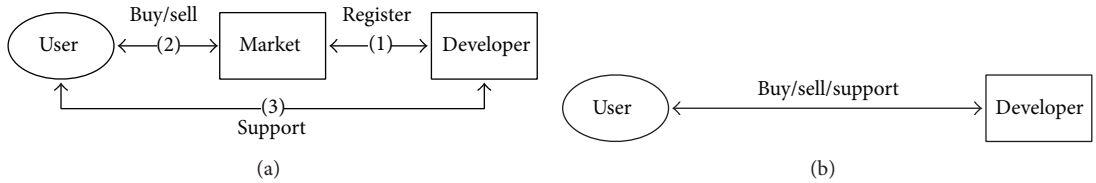


Figure 1: Traditional distribution—(a) individual markets; (b) direct from developers.

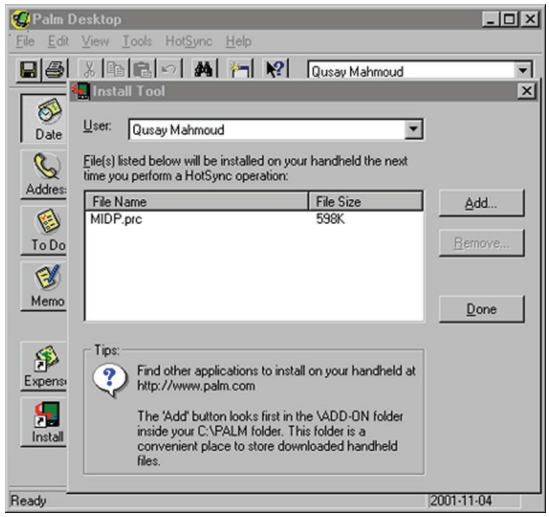


Figure 2: Application management in Palm OS. Legacy devices needed to connect to a PC to install mobile applications.

application from either (a) individual markets or (b) developers directly, as shown in Figure 1. When digital download services were introduced, users could still purchase from either source and receive support for that application by directly contacting the developer.

Conventional markets, however, must address the following two issues: *license management* and *software installation*. For license management, users purchasing from a web-market could download the application directly from the specific website. For authorization, users received license codes through emails or receipts. Users had to keep or request new license codes from distributors when they were required to reinstall the app.

The mobile handset market, in addition to the PC market, also provided digital download services. For example, legacy mobile devices, such as those based on *Palm OS* and *Windows CE*, widely used until the late 2000s, enabled users to install any application they chose to their devices.

The installation process, however, was not convenient. Figure 2 shows an example of installing an application on a *Palm OS*-based device. To install an application, users had to manage a desktop application that synchronized with the mobile device.

Although later Wi-Fi-enabled devices could download and install applications without desktop tools, the purchase and authorization process remained the same as shown in Figure 1. Users still managed their license codes themselves.

2.1. Online Application Stores. Online application stores (OASs) provide users with easier license and application code management. When a user purchases an app from an OAS, it requires only one click to install, reinstall, or update the app. In fact, the market share of *Palm OS*, *Windows CE*, and even *Symbian OS* quickly decreased once Apple launched the *Appstore* for *iPhone*. OASs are not only used with mobile devices. PC environments, including Windows and Mac OS X, and software distributors such as Amazon are rapidly deploying market services, as shown in Figure 3.

Multiple OASs are often preinstalled or installed by users on their devices and systems. By connecting to an OAS, a user can easily purchase applications. When a user needs support, they can easily get updates from their application provider, as shown in Figure 4.

2.2. Types of OAS User Registration. We separate OASs into the three groups discussed in [1]: OAS from OS holder (Type 1), OAS from Content distributor (Type 2), and OAS from mobile carriers (Type 3).

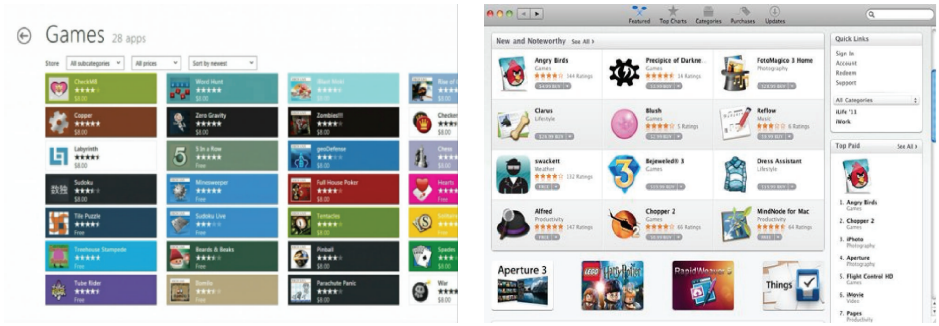
(i) Type 1: the store application is preinstalled on the device. Users register their accounts with the service. To purchase applications, users must also register their billing information. Depending on the billing information or location information, the store can provide localized services. Microsoft's *Windows Store*, Apple's *Appstore*, the Ubuntu *Software Center*, and *Google Play* are examples.

(ii) Type 2: users manually install the store application on their device. Users register their accounts. To purchase applications, users must also register their billing information. Stores authenticate users with the billing information. *Amazon Appstore* and *Steam Online* are examples.

(iii) Type 3: mobile carriers/manufacturers preinstall their own OASs on the device. Mobile users are already registered through their carrier subscription information. The store verifies the information from the USIM in the device. Only subscribed devices can use the service. Users must be connected to the cellular network.

3. Application Management from Multiple OASs

In this section, we discuss issues with application management from multiple OASs and extend the PAS introduced in [1] to overcome such issues.



(a) (b)

Figure 3: (a) Windows marketplace (b) Mac Appstore.

Figure 4: Online application distribution.

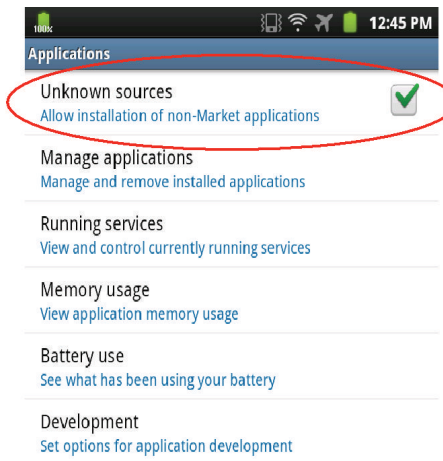


Figure 5: Android OS allows unauthorized applications.

3.1. Application Management from Multiple OASs

3.1.1. Software License Check Problem. Software piracy has long been a serious security problem. Many proposals [2–4] have attempted to prevent this problem by the use of a software license confirmation. They focus on the verification of the validity of the software license from the original source.

For example, the Android system enables anyone to develop and distribute Android software. It uses the Android application package file (APK) format to distribute and install applications and middleware to the device. Although the Android system initially demands that all applications be signed by the application developer to ensure the trust of the applications, installing unauthorized applications manually is also allowed, as shown in Figure 5. This openness permits the illegal distribution of cracked applications and malware (Graham Cluley, "Android malware poses as Angry Birds Space game," NakedSecurity, SophosLab April 12, 2012) to the Android system [5].

Therefore, many researchers have focused on prevention mechanisms against such threats [6–9], including online authentication. Such authentication systems, however, can decrease the availability of applications.

While many vendors do deploy online authentication systems, Type 3 distributors generally use an authentication process via a wireless network. For example, Korean local distributors, including *SKT T-store* and *KT Olleh market*, commonly utilize users' subscribed information in the USIM over the cellular network. When a user launches an application, they must be connected to the network. Those who are not connected to the network via a cellular connection fail to be authenticated.

3.1.2. Software Support without Original OAS. In the traditional application distribution environment shown in Figure 1, stores only provide applications to users. Users then contact the developers directly for support. Although this is not a very convenient process, once a user has purchased an application, they can obtain continued support from the developer.

In contrast, in the current distribution process shown in Figure 4, OASs not only sell applications but also provide support to users. Although such a mechanism brings huge convenience to users, they must maintain their connection to the store to receive this support. A user who cannot contact the store will fail to get further support and must purchase the same application again, from a different store that he can contact. This generally occurs for Types 2 and 3 cases, where the OAS only provides service for the localized domain.

3.1.3. OAS Management Problem. OAS users can purchase software from multiple OASs, as shown in Section 2.2. This can cause problems with verifying the license. Although allowing multiple OASs in a device allows users to choose their preferred service, any OAS that a user contacts can manage the software. Whereas legacy distribution systems allow users to get software support, regardless of where the application was purchased, OAS users can only get support from the OAS from whom they made their purchase.

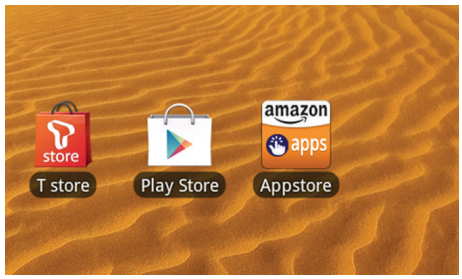


Figure 6: An example of various OASs in one mobile device: *SKT T-store* (Type 3), *Google Play store* (Type 1), and *Amazon Appstore* (Type 2).

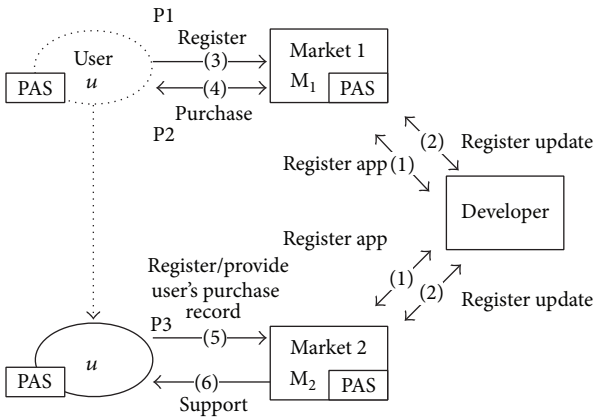


Figure 7: System model.

Therefore, OAS users who purchase apps from multiple OASs must manage multiple OAS systems in the device, as shown in Figure 6. This increases the management overhead, especially to mobile device users.

3.2. PAS Model. In order to resolve the issues discussed above, we propose a PAS model. This enables users who have already purchased applications to receive support when they change their status or cannot reach the original OAS, temporarily or permanently [1]. We define the PAS as a trusted entity that stores users' purchase records. We have limited the functionality of the PAS model to the mobile environment.

4. Improved PAS Model

Maintaining an additional trusted entity for the PAS could increase the management overhead. In this paper, therefore, we extend the model by adding a PKI feature and modify the PAS as a part of this service. This does not require an additional entity, and hence there are no additional management issues. When a user purchases an application from OAS_1, OAS_1 stores the user's purchase record. At a later time, if a user loses contact with OAS_1, he can obtain support from OAS_2 by providing this proof of purchase. We assume that a user is always registered with at least one service.

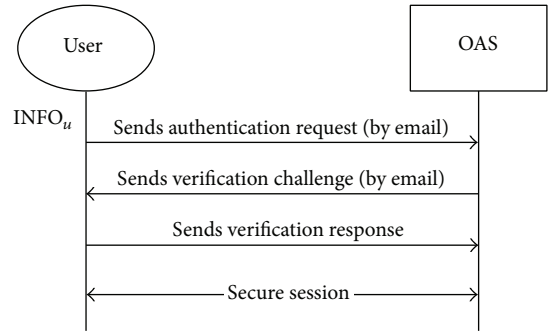


Figure 8: Checking *UAddr* by email verification.

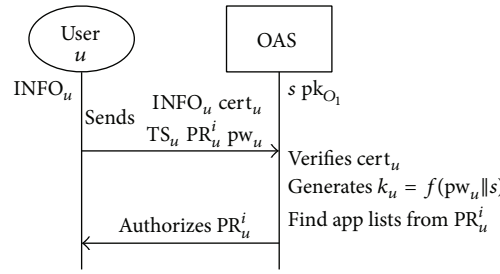


Figure 9: P1: Registration phase.

4.1. Improved System Model. We assume the following system model, illustrated in Figure 7. A developer provides applications to the OASs. A user, U, purchases applications from the OASs. The OASs share their public keys, pk_j, $0 \le j \le n$, where n is the ID of the OAS. Stores (OAS_1 and OAS_2) have a secure association. OASs also share the seed secret, s. Developers always register (1) and update (2) their applications. U must first register himself to an OAS, say, OAS_1. (3) After successful registration, U may purchase multiple applications from OAS_1. (4) OAS_1 stores U's information using the PAS. When U has a status change, he may request to update his registration to a newly connected store, OAS_2 (5). OAS_2 provides services after validating U (6).

4.2. P1: Initial User Registration Phase. The user registration process is initiated when a user first registers with a specific store. Let a user, U, register with store OAS_1.

When U requests their registration to OAS_1, OAS_1 establishes a secure channel with U. We assume that email is used to establish the secure channel, as is used by many Internet services. Figure 8 shows an example of this process.

The Registration phase, P1 in Figure 7, registers U to OAS_1 as shown in Figure 9. $\mathrm{INFO}_{\mathcal{U}}$ denotes the user purchase record stored by the OAS. When a user requests support from OAS_1, the store verifies $\mathrm{INFO}_{\mathcal{U}}$. $\mathrm{INFO}_{\mathcal{U}}$ includes the elements in Table 1.

When U and OAS_1 establish a secure channel, U selects and sends PWD to OAS_1. OAS_1 gathers $\mathrm{INFO}_{\mathcal{U}}$ and generates $\mathrm{cert}_{\mathcal{U}}$ as follows:

$$\mathrm{cert}_{\mathcal{U}} = \mathrm{sign}_{sk_{O_1}} \left\{ h\left(\mathrm{Addr}_{\mathcal{U}}^{P}\right) \| h\left(\mathrm{pw}_{\mathcal{U}}\right) \| \cdots (\mathrm{opt}) \cdots \| h\left(\mathrm{TS}\right) \| h\left(\mathrm{PR}_{\mathcal{U}}\right) \right\}, \quad (1)$$

Table 1: $\text{INFO}_{\mathcal{U}}$ elements.

Element	Description
$\text{Addr}_{\mathcal{U}}^{P}$	User's primary contact information (e.g., email address)
$\text{Addr}_{\mathcal{U}}^{S}$	User's supplementary information (e.g., phone number; optional)
$\text{PN}_{\mathcal{U}}$	Device information (optional)
$\text{CN}_{\mathcal{U}}$	Country code (optional)
$\text{CR}_{\mathcal{U}}$	Carrier code (Type 3)
TS	Timestamp
$\text{pw}_{\mathcal{U}}$	Passcode for the registration
$\text{PR}_{\mathcal{U}}$	Purchase record of U

where sk_{O_1} is a private key of OAS_1 and $\text{sign}_k\{m\}$ denotes a signature of m signed by k. $\text{PR}_{\mathcal{U}} = \text{enc}_{k_{\mathcal{U}}}\{\text{App}_i\}$, where i is the purchased app ID and $k_{\mathcal{U}} = f(\text{pw}_{\mathcal{U}}||s)$, where $f(m)$ is a key generation function with input m and s is the seed secret of the OASs. opt denotes optional information for deployment.

OAS_1 then sends TS, $\text{PR}_{\mathcal{U}}$, and $\text{cert}_{\mathcal{U}}$ to U. U stores $\text{cert}_{\mathcal{U}}$, $\text{PR}_{\mathcal{U}}$, and TS.

4.3. P2: Purchase Phase. The purchase phase, P2 in Figure 7, is invoked when U purchases applications from OAS_1.

When U purchases App_i from OAS_1, OAS_1 updates $\text{PR}_{\mathcal{U}}$. In the first step, OAS_1 decrypts $\text{PR}_{\mathcal{U}}$ with $k_{\mathcal{U}}$ and adds APP_i to the application list. If s is updated to s^{new}, OAS_1 generates a new $k_{\mathcal{U}}^{\text{new}} = f(\text{pw}_{\mathcal{U}}||s^{\text{new}})$. Then, $\text{PR}_{\mathcal{U}}^{\text{new}}$ is generated by encrypting the updated application list using $k_{\mathcal{U}}^{\text{new}}$.

Finally, OAS_1 sends $\text{PR}_{\mathcal{U}}^{\text{new}}$ to U, where it is also stored.

4.4. P3: Purchase Authentication Phase. The purchase authentication phase, P3 in Figure 7, is invoked when U contacts a new OAS, one from whom he did not purchase the application. We consider the case where U requests support from OAS_2. We assume that U registers himself with OAS_2, using the user registration phase, or temporarily contacts OAS_2 and then requests support for an application already purchased from OAS_1.

4.4.1. Step 1: Check User's Registration Information. To verify U's purchase record, OAS_2 checks U's registration information $\text{Addr}_{\mathcal{U}}$, as shown in Figure 8. Through a secure channel, U requests the purchase authentication from OAS_1 and sends $\text{INFO}_{\mathcal{U}}$ with $\text{pw}_{\mathcal{U}}$, $\text{cert}_{\mathcal{U}}$, $\text{TS}_{\mathcal{U}}$, and $\text{PR}_{\mathcal{U}}$ to OAS_2. OAS_2 then verifies $\text{cert}_{\mathcal{U}}$ with OAS_1's public key pk_{O_1}. After verifying U's registration information, OAS_2 generates $k_{\mathcal{U}}$ with $\text{pw}_{\mathcal{U}}$ and s. OAS_2 then decrypts U's purchase record, $\text{PR}_{\mathcal{U}}$, with $k_{\mathcal{U}}$.

4.4.2. Step 2: User Authorization. The processes are slightly different depending on the user's status. In this paper, we show two cases: U temporarily uses a roaming service and U permanently changes his OAS.

Case A: U Temporarily Uses a Roaming Service. When U connects to OAS_2 as a roaming service, OAS_2 grants temporary authorization to U. U still has his original $\text{PR}_{\mathcal{U}}$, $\text{INFO}_{\mathcal{U}}$, and $\text{Cert}_{\mathcal{U}}$, and OAS_2 does not send a new certificate. OAS_2 has access to the billing information of U and can request payment. The authorization remains valid for a specific time period; for example, OAS_2 can authorize U for one day.

Case B: U Permanently Changes His OAS. When U permanently changes from OAS_1 to OAS_2, $\text{Cert}_{\mathcal{U}}$ from OAS_1 is revoked and OAS_2 issues a new $\text{Cert}_{\mathcal{U}}^{\text{new}}$ to U. $\text{INFO}_{\mathcal{U}}$ is updated. For example, U may have a new $\text{Addr}_{\mathcal{U}}^{S}$. If the user keeps his old email, $\text{Addr}_{\mathcal{U}}^{P}$ does not change. If the user connects using the same device, $\text{PN}_{\mathcal{U}}$ remains unchanged. $\text{TS}_{\mathcal{U}}$ is updated. U receives $\text{INFO}_{\mathcal{U}}^{\text{new}}$, $\text{Cert}_{\mathcal{U}}$ and stores them with $\text{PR}_{\mathcal{U}}$. OAS_2 also stores the information. By this process, OAS_2 can bill U.

5. Security Analysis

In this section, we show that the security of the design satisfies standard security requirements and also show that the design is secure against possible attack. We assume OAS_i, where i is the ID of the OAS, can be trusted. Performance is not an issue in this paper and depends upon the actual deployment case.

5.1. Security Requirements. The following are the security requirements for the PAS model.

(i) Nonrepudiation: the user should not be able to claim that his records are invalid.

(ii) Authentication: the distributor must be able to validate the user's request.

(iii) Privacy: the distributor can only know the user's information after the user is approved.

The handling of malicious applications in the store is not the focus of this paper.

5.2. Nonrepudiation. $\text{pw}_{\mathcal{U}}$ is chosen by U, and U does not know the k_U generated from $\text{pw}_{\mathcal{U}}$. Since OAS_2 can request information about $\text{PR}_{\mathcal{U}}$ only when U requests a service, repudiation from U can be prevented.

5.3. Authentication. $\text{Cert}_{\mathcal{U}}$ enables OAS_i to check the validity of U. Since only a valid OAS_i can generate a $\text{Cert}_{\mathcal{U}}$, using the private key sk_{O_i}, a malicious user or other attacker cannot forge or abuse it.

5.4. Privacy. Without $\text{pw}_{\mathcal{U}}$, OAS_i cannot access the application list in $\text{PR}_{\mathcal{U}}$. OAS_i can only generate the $k_{\mathcal{U}}$ that decrypts $\text{PR}_{\mathcal{U}}$ to see the application list when U sends $\text{pw}_{\mathcal{U}}$.

U can replace $\text{pw}_{\mathcal{U}}$ at any time. Although a specific OAS_i can see the application list if a user chooses to use a temporary roaming service, it will not be aware of any future changes to $\text{pw}_{\mathcal{U}}$.

Only hashed user information from Table 1 is stored in $\text{Cert}_{\mathcal{U}}$. Thus, an unapproved OAS_i cannot know the information before U provides them with access.

5.5. Security against Possible Attack Scenarios. We assume that a malicious user Eve, E, could try to obtain support from a market without any purchase record. E could try the following scenarios.

5.5.1. Fraudulent User Tries to Get Authorization Illegally. E impersonates U. In this case, where the attacker impersonates a legal user, E would require $\text{INFO}_{\mathcal{U}}$, including $\text{Addr}_{\mathcal{U}}$, to impersonate U in OAS_1 or OAS_2.

E would not, however, be able to enter $\text{Addr}_{\mathcal{U}}$ during the PAS registration phase described in Section 4.4.1. Securing U's email account is not the focus of this paper.

Even when E compromises U's device and extracts $\text{INFO}_{\mathcal{U}}$ and $\text{Cert}_{\mathcal{U}}$, E still does not know the password $\text{pw}_{\mathcal{U}}$. Deploying the PAS model, OAS_i can limit the number of password attempts. For example, if several invalid password entries are attempted, OAS_i can temporarily place U's account on hold. In such a case, U would have to contact the service by phone or physical mail. We do not show the details of this process in this paper.

5.5.2. Forged Purchase Record. The malicious user E could forge his own purchase record $\text{PR}_{\mathscr{E}}$. In this case, E would have to be able to modify $\text{PR}_{\mathscr{E}}$ in the PAS. For this to succeed, E would have to know s in order to generate $k_{\mathscr{E}}$ and then generate $\text{Cert}_{\mathscr{E}}$. This is impossible without knowing sk_{OAS_i}.

6. Conclusion

With OASs becoming the main channel of software distribution, software license authentication issues present a potential problem when using multiple OASs. We have discussed possible issues from using multiple OASs and proposed an improved PAS model that reduces management overheads without any additional entity, while still allowing users to obtain support from multiple OASs. We refined our model to support a temporary roaming situation, as well as a permanent OAS change. We described the security of the proposed model.

Our design shows not only the technical availability of ongoing benefits to users, but also a possible business model for OASs.

Conflict of Interests

The authors declare that there is no conflict of interests regarding the publication of this paper.

Acknowledgments

This research was supported by the Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education, Science, and Technology (no. 2012R1A1A1010667).

References

[1] K. Han and T. Shon, "Authentication of mobile applications through various local distributors," *Multimedia Tools and Applications*, 2013.

[2] K. Fukushima, S. Kiyomoto, and Y. Miyake, "Software protection combined with tamper-proof device," *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, vol. E95.A, no. 1, pp. 213–222, 2012.

[3] G. Horvat, D. Sostaric, and D. Zagar, "Multi-agent based software licensing model for embedded systems," in *Agent and Multi-Agent Systems. Technologies and Applications*, G. Jezic, M. Kusek, N.-T. Nguyen, R. Howlett, and L. Jain, Eds., vol. 7327 of *Lecture Notes in Computer Science*, pp. 648–657, Springer, Berlin, Germany, 2012.

[4] W. Liu, "Software protection with encryption and verification," in *Software Engineering and Knowledge Engineering: Theory and Practice*, Y. Wu, Ed., vol. 115 of *Advances in Intelligent and Soft Computing*, pp. 131–138, Springer, Berlin, Germany, 2012.

[5] M. Backes, S. Gerling, and P. von Styp-Rekowsky, "A novel attack against android phones," http://arxiv.org/abs/1106.4184.

[6] P. Albano, A. Castiglione, G. Cattaneo, and A. de Santis, "A novel anti-forensics technique for the android OS," in *Proceedings of the 6th International Conference on Broadband and Wireless Computing, Communication and Applications (BWCCA '11)*, pp. 380–385, October 2011.

[7] F. di Cerbo, A. Girardello, F. Michahelles, and S. Voronkova, "Detection of malicious applications on android OS," in *Proceedings of the 4th International Conference on Computational Forensics (IWCF '10)*, pp. 138–149, Springer, Berlin, Heidelberg, 2011.

[8] W. Enck, D. Octeau, P. McDaniel, and S. Chaudhuri, "A study of android application security," in *Proceedings of the 20th USENIX Conference on Security (SEC '11)*, p. 21, USENIX Association, Berkeley, Calif, USA, 2011.

[9] W. Zhou, Y. Zhou, X. Jiang, and P. Ning, "Detecting repackaged smartphone applications in third-party android marketplaces," in *Proceedings of the 2nd ACM Conference on Data and Application Security and Privacy (CODASPY '12)*, pp. 317–326, ACM, San Antonio, Tex, USA, 2012.