A Software Security Optimization Architecture (SoSOA) and Its Adaptation for Mobile Applications

https://doi.org/10.3991/ijim.v15i11.20133

Amr Abozeid
Jouf University, Jouf, Saudi Arabia
Al-Azhar University, Cairo, Egypt

AbdAllah A. AlHabshy (✉), Kamal ElDahshan
Al-Azhar University, Cairo, Egypt
AbdAllah@Azhar.edu.eg

Abstract—Security attacks become daily news due to an exposure of a security threat in a widely used software. Taking software security into consideration during the analysis, design, and implementation phases is a must. A software application should be protected against any security threat such as unauthorized distribution or code retrieval. Due to the lack of applying a software security standard architecture, developers may create software that may be vulnerable to many types of security threats. This paper begins by reviewing different types of known software security threats and their countermeasure mechanisms. Then, it proposes a new security optimization architecture for software applications. This architecture is a step towards establishing a standard to guarantee the software's security. Furthermore, it proposes an adapted software security optimization architecture for mobile applications. Besides, it presents an algorithmic implementation of the newly proposed architecture, then it proves its security. Moreover, it builds a secure mobile application based on the newly proposed architecture.

Keywords—Reverse engineering, Security architecture, Security optimization, Software protection, Source code protection

1 Introduction

Developers afford a great work in order to produce a single application. So, securing this hard work is absolutely necessary; i.e. any software application should be protected. Protecting any software application - includes but not limited to - the following:

1. Secure source code against source code hacking, and code reverse engineering.
2. Secure software applications against illegal download and illegal copy and/or distribution.
3. Secure data against internal threats (data loss) and external threats (Information leakage).

To secure data against internal threats, then the countermeasures are including but not limited to data protection (data loss prevention), utilize security frameworks and libraries, and error monitoring and handling exception. To secure data against external threats, then the countermeasures are including but not limited to security tests, data validation, access control (authorization), authentication, logging and intrusion detection, and SQL injection prevention.

Developers should take software security into account during the analysis, design, and implementation phases. To guarantee the software security, developers should follow a software security standard architecture. Besides introducing new algorithms that guarantee digital rights management (DRM), this paper integrates all security mechanisms into a single robust software security architecture. This paper presents a secure optimization architecture for software security. Then it presents an adapted architecture for mobile applications. By using the proposed architecture, software developers can integrate all security mechanisms in-house. Accordingly, they do not have to send the source code to a respectable software security company to add the needed security mechanisms.

The rest of this paper is organized as follows. Section 2 presents the mechanisms that may be used to secure software applications. Section 3 proposes a new software security optimization architecture (SoSOA). Section 4 adapts SoSOA to mobile applications. Finally, Section 5 presents the conclusion and further works.

2 Software Applications Security

This section reviews the mechanisms used to ensure the security of any software application. These mechanisms could be used to secure source code, software application and data.

2.1 Secure source code

A software package may have several alternatives. One of them implements an original algorithm and the others just imitate it. If the programmers can imitate other programmers work, then the computer may imitate a given application. The programmer/computer captures the main features of the original software package and builds a new one (using what is called reverse engineering) [1].

Source code hacking: To prevent an attacker form stealing the source code, developers should use at least one of the following methods:

1. Encrypt source code files [2, 3]
2. Add watermarks to source code via metadata [4-6]
3. Use a software package for code protection such as Arxan EnsureIT, StarForce C++ Obfuscator, etc [7].

Code reverse engineering: One of the most efficient countermeasures to reverse engineering that the community seems to be agreed upon is code obfuscation[8-12].

Some of the most popular obfuscation tools are ProGuard [13], DexProtector [14], and DexGuard [15]. These tools use cryptographic algorithms besides security checks in Android APK to prevent attacks such as intrusions. Once these tools detect that the code is being hacked, the application will be blocked immediately.

2.2 Secure software application

This section presents DRM functionality; security mechanisms that could be used to prevent illegal software application download and/or illegal copy and illegal distribution.

Secure software application: against illegal download. If the program is free, then there is no illegal download. Otherwise, A digital distribution platform (Such as app store, play store, galaxy apps, amazon app store, etc.) could be used to handle such a problem [16]. The illegal copied-software application should not work.

Secure software application: against illegal copy and/or illegal distribution. This can be done using a dongle [17] and/or code signing [18] with Android Application Licensing [7].

2.3 Secure data

This section discusses the mechanisms used by developers to secure data against both internal and external threats.

Internal threats (Data Loss). Data loss means losing sensitive information due to human errors, computer viruses, adversaries' attacks, computer hardware or software failure, or natural disasters. Various mechanisms utilized to protect stored and transmitted data such as Utilize Security Frameworks and Libraries [19, 20], Data Protection (Data Loss Prevention - DLP), Error Monitoring and Handling Exception.

External threats (Information leakage): A software application could leak sensitive information such as application configuration details, comments within the code or personal data. This information could be used to attack the software application, its network host or its users. Various security mechanisms utilized to prevent information leakages, such as Authentication [21, 22], Access Control (Authorization), Logging and Intrusion Detection [23, 24], Auditability, SQL Injection [25-28], Data Validation, and Security Tests [29, 30].

3 Newly Proposed Software Security Optimization Architecture

This section proposes a new security architecture which can be used by developers in order to protect their software application. This architecture could be described as follows:

1. During the application development phase, the developer should secure the software applications using:

   a) Mechanisms that are used to ensure source code protection and software application protection as described in Sections 2.1 and 2.2 respectively. Accordingly, developers could:7
      i. Add a watermark to the source code and encrypt its files, in order to avoid source code stealing.
      ii. Use obfuscation to prevent reverse engineering. Furthermore, for Android applications, tools such as ProGuard, DexProtector and DexGuard could be used, in order to avoid reverse engineering.
   b) Mechanisms that are used to prevent data loss and information leakage as described in Section 2.3.

2. Assure legal and paid software application utilization: Software application should be published on app store and this app store manages such issues.
3. Secure software application against illegal distribution: To protect a software application against illegal distribution, developers should use a DRM system.

3.1 Newly proposed secure software applications against illegal distribution

Whenever the developers do not want to use an external DRM system, they may use the following new proposed DRM procedure.

The notations to be used in the proposed procedure may be described as follows:

- H: is a secure hash function.
- MAC: is a secure message authentication code (a keyed secure hash function).
- $\|$: is the concatenation operation.
- Nonce: is a random 256 bits string that may be used only once.
- CMK: is the Company's Master Key, which is a 256 bits secure company's key for a specific software application.
- MSN: is the machine motherboard serial number.
- $IMEI$: is the International Mobile Equipment Identifier, which is also known as the Mobile Equipment Identifier ($MEID$).
- $UserID$: is the User identification, which may be the user's email, phone number, fax number, or mobile phone number.
- $MAC - KEY$: is the key that is used to generate the MAC such that $MAC - KEY = H(Nonce \parallel CMK \parallel MSN)$.
- $app - KEY$: is a product key such that, app-$KEY = H(UserID \parallel MAC - KEY \parallel MSN)$.
- $app - KEY - file$: is a file that contains the $app - KEY$.
- $MAC[app - KEY - file]$: is the value of applying H on the $app - KEY - file$ such that $MAC[app - KEY - file] = H(MAC - KEY \parallel app - KEY - file)$.

To avoid sending the source code to a third party to add the DRM functionalities, the developers may use the following procedure:

Install and activate the application: To install the application, the software development company and the user should do as follows:

1. The company server deploys the software application on an app store.
2. The user:

   a) Buys the software application from a trusted app store. The purchase-ticket containing the $UserID$ and MSN should be signed.
   b) Downloads the purchased software application from the app store.
   c) Installs the software application and opens it for the first time, then registers as a new user.
   d) Sends $UserID$, MSN and purchase-ticket via a secure channel to the company's server.

3. The company's server:

   a) Ensures the application purchase for this $UserID$ and MSN.
   b) Generates the $app - KEY$ for this specific user.
   c) Creates the $app - KEY - file$.
   d) Generates the $MAC[app - KEY - file]$.
   e) Signs the $MAC[app - KEY - file]$ using the digital signature of the company's server.
   f) Stores the values of $(UserID, MSN, app - KEY, Sign\ of\ MAC[app - KEY - file])$ in a protected database.
   g) Sends $MAC - KEY$, $app - KEY - file$ and Sign of the $MAC[app - KEY - file]$ to the user via a secure channel.

4. The user completes the installation as follows:

   a) Uses the $app - KEY - file$ to activate the application.
   b) Stores and protects the values of the $MAC - KEY$, and the Sign of the $MAC[app - KEY - file]$.

At the application's start-up: At start-up, the application should check the machine's legitimacy by:

1. Generating a new $app - KEY - file$ using the MSN of the current machine.
   Constructing a new $MAC[app - KEY - file]$.
   Constructing the old $MAC[app - KEY - file]$ from the signed $MAC[app - KEY - file]$.
   Checking whether the value of the old $MAC[app - KEY - file]$ equals the value of the new one. If yes, then run the software application. Else, abort running the software application.

Whenever the app starts up and the machine connected to the Internet: Immediately after the device is connecting to the internet, the company's server should ensure that the app-KEY-file has not been altered. This should be done, even if the software application is opened and being used. To this end, the subsequent steps should be followed:

1. The application installed on user's machine:

   a) Generates a new $app - KEY - file$ using the MSN of the current machine and then constructs a new $MAC[app - KEY - file]$.
   b) Sends $UserID$, MSN, and the new $MAC[app - KEY - file]$ to the company's server via a secure channel.

2. The company's server checks whether the $UserID$ is registered. If yes, then constructs a signed $MAC[app - KEY - file]$ using the received $MAC[app - KEY - file]$ and sends it to the user using a secure channel. Else, the company's server revokes the operation.

After receiving the new signed $MAC[app - KEY - file]$, the software application checks whether it is the equals the old signed $MAC[app - KEY - file]$. If yes, then the software application runs. Else, the software application aborts.

Whenever the mobile is lost, changed or destroyed: A new product key should be generated for the legitimate user. To this end, the subsequent steps might be followed:

1. The user:

   a) Downloads the application from an authentic app store.
   b) Installs the software application and opens it for the first time, then requests a new activation.
   c) Sends $UserID$ and the new MSN to the company's server via a secure channel.

2. The company's server checks whether the $UserID$ is registered. If yes, then it sends a validation message to this user using the $UserID$. Else, it revokes the operation.
3. After receiving the validation message, if the user has asked for a new activation, then sends a response to the validation message with the current machine MSN. Else, the user revokes the operation.
4. After receiving the response, the company's server checks the user legitimacy. If the user is not legitimate, then aborts the operation. Else, the company's server:

   a) Revokes the old product key associated with this mobile number.
   b) Creates a new product key $app - KEY$ for this specific $UserID$ and the new MSN and sends $MAC - KEY$, $app - KEY - file$ and the signed $MAC[app - KEY - file]$ to the user via a secure channel.

5. The user completes the installation as follows:

   a) Uses $app - KEY - file$ to activate the application.
   b) Stores and protects the values of $MAC - KEY$, and Sign of the $MAC[app - KEY - file]$.

Figure 1 shows the newly proposed software security optimization architecture.

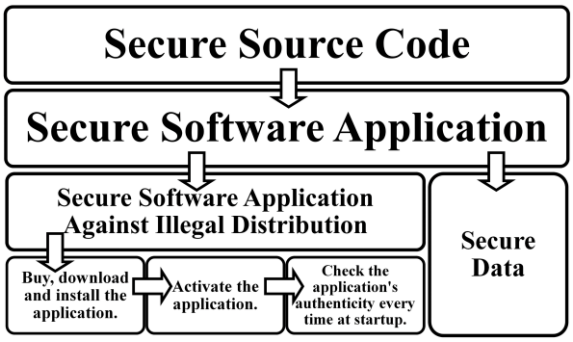

Fig. 1. Software secure optimization architecture

3.2 Security analysis

This section provides a security analysis for the newly proposed DRM procedure to prove its ability to prevent all famous attacks' scenarios.

First scenario: If the attacker Eve tries to register as a new user without a purchase-ticket, then the company's server will revoke the operation. Also, since every purchase-ticket contains someone's $UserID$ and MSN, then Eve never be able to use it.

Second scenario: Since all communications use a secure channel, then when a legal user tries to register for the first time, the attacker Eve cannot change the couple $(UserID, MSN)$ to her own in order to get the activation parameters. Moreover, if Eve succeeds to change the couple to her own data, then she cannot change it within the purchase-ticket. Consequently, the company's server will revoke the operation.

Third scenario: Since the activation parameters are produced for a specific machine, then the attacker Eve cannot capture them to complete the installation and activate the application.

Forth scenario: Whenever the attacker Eve tries to ask for a new activation, the company's server will revoke the operation. This is because her $UserID$ does not exist in the company's server database. Moreover, if Eve asks for a new activation with a legal $UserID$, then the legal user will be acknowledged and consequently revoke the operation. And even if the legal user responded positively, the new activation parameters will be produced for the legal user's machine, not for Eve's machine.

Fifth scenario: When a legal user asks for a new activation, the license on the prior machine will be revoked. As the application will not work once the prior machine is connected to the Internet. So, the legal user will never to be able to have the application installed on more than one machine at a time.

Sixth scenario: An attacker Eve might buy an old machine with an installed application. As mentioned, the application checks the machine authenticity when it connects to the Internet. Thus, the application will revoke the old license when the legal user asks for a new activation. Consequently, the application will stop running on the old machine when it connects to the Internet.

3.3 Efficiency analysis

The following notations are used to analyze the performance of the newly proposed DRM procedure:

1. T_{MAK}: is the time complexity of the used message authentication code.
2. T_{GAKF}: is the time complexity of generating the app_Key_file.
3. T_{sign}: is the time complexity of the used digital signature.

Accordingly, the time complexity required to install and activate the application is $T_{MAK} + T_{GAKF} + T_{sign}$. The time complexity requited at the application's start-up is $T_{MAK} + T_{GAKF}$. The time complexity required whenever the app starts up and the machine connected to the Internet is $2T_{MAK} + T_{GAKF}$. The time complexity required whenever the mobile is lost, changed or destroyed is $T_{MAK} + T_{GAKF} + T_{sign}$.

Since the newly proposed DRM procedure just sends and receives a few messages with negligible time according to the current Internet's speed. Besides, it performs at most three security mechanisms such as MAC and digital signature with trivial time according to the modern computers' speed. Consequently, the efficiency of the newly proposed DRM procedure is similar to the efficiency provided by Google or any other companies DRM procedures.

3.4 A comparative study among various related security architectures

This section provides a comparative study among the proposed SoSOA, and various related security architectures (Android applications licensing) that provide DRM functionality such as Google play licensing, Amazon DRM, Android licensor, Droid activator, etc [7].

The proposed SoSOA is an open source architecture for software security optimization. It enables software developers to add DRM functionality in-house as described in Section 3.1. The DRM functionality will be adapted in Section 4.1 for mobile applications. Furthermore, the proposed SoSOA urges software developers to secure the source code and the data using the mechanisms described in Sections 2.1 and 2.3 respectively. Besides, the proposed SoSOA, is a platform-independent software security architecture.

Google play licensing is an open source system. that provide digital rights management (DRM) functionality. Even though, Google play licensing requires having Google accounts for both developers and users. Also, it requires source code to add the protection mechanisms. Moreover, the copy protection mechanism is no longer supported. Furthermore, it does not support Java/C/C++ code protection.

Amazon DRM provides DRM functionality. Even though, it requires source code to automatically add the protection mechanisms, which produce unexpected results. For example, sometimes verification errors cause apps to close. Moreover, it does not support Java/C/C++ code protection. Besides, it is a proprietary system.

Android licenser is an open source system that provides DRM functionality. Even though, it requires source code to add the protection mechanisms. Also, it provides low level of protection. Moreover, an adversary could bypass the protection due to using

public channels to deliver the keys. Furthermore, it does not support Java/C/C++ code protection. Besides, it is a paid DRM service.

Droid activator is an open source system. Even though, the developer needs to deploy his/her own server, for the sake of storing the server part of Droid activator protection. Also, it provides a low level of protection. Moreover, it is easy to capture the activation keys. Furthermore, it does not support Java/C/C++ code protection.

Unfortunately, these Android applications licensing provide neither source code protection nor data protection. Moreover, software developers have to provide their own application source code to a software security company to add DRM functionality. Thus, the proposed SoSOA is superior to the Android applications licensing, as it provides source code protection, DRM, and data protection functionalities in-house.

Table 1 summarizes the comparative study, which presents the results of the comparison among the proposed SoSOA and various Android applications licensing.

Table 1. A comparative study summrization

Functions	Google Play Licensing	Amazon DRM	Android Licenser	Droid Activator	SoSOA
Open Source System	Yes	No	Yes	Yes	Yes
DRM Functionality	Yes	Yes	Yes	Yes	Yes
In-house DRM Functionality	No	No	No	No	Yes
High Level Protection	Yes	Yes	No	No	Yes
Source Code Protection	No	No	No	No	Yes
In-house Source Code Protection	No	No	No	No	Yes
Data Protection	No	No	No	No	Yes

4 An Adapted Software Security Optimization Architecture for Mobile Applications (SoSOA-MA)

To adapt SoSOA to mobile applications, $UserID$ and MSN will be replaced by the $Mobile\ Phone\ Number$ (MPN) and $IMEI$ respectively. To secure software mobile app against illegal distribution, this section presents an applied algorithmic methodology for the adapted architecture. It also presents an Android implementation for the proposed DRM algorithms in Section 4.1. Moreover, it presents a case study (VirTour application) in Section 4.2.

4.1 An applied methodology for securing mobile applications against illegal distribution

This section presents the algorithms that may be used to secure mobile applications against illegal distribution after deploying it on an app store.

Download, install and activate a mobile application: Once a user buys the mobile application and gets the purchase-ticket, the newly proposed algorithm 1 may be used to complete legal installation of the mobile application. This algorithm generates the activation parameters for a specific mobile phone. Algorithm 1 creates $MAC - KEY$,

$app - KEY - file$, $Signed - MAC[app - KEY - file]$ for a particular user and stores them in a database. Meanwhile, the company's server delivers these values to the user via a secure channel. After getting these values, the app on the mobile device should protect them from being altered. After that, the app will generate and use the $app - KEY - file$ for its activation. Then, the app constructs $MAC[app - KEY - file]$, which cannot be altered. This algorithm is activated only once at application's installation.

Algorithm 1 Install and activate a Mobile Application

Input: MPN, IMEI, purchase-ticket
Output: Complete installation of a mobile app for a specific mobile
Begin
procedure GENERATE ACTIVATION PARAMETERS AT THE SERVER (MPN, $IMEI$, purchase-ticket)
    if valid purchase-ticket == false **then**
        Abort the operation.
    Else
        Generate the $app - KEY$ for this specific user.
        Create the $app - KEY - file$.
        Generate the $MAC[app - KEY - file]$.
        Sign the $MAC[app - KEY - file]$.
        Store the values of (MPN, $IMEI$, $app - KEY$, Sign of $MAC[app - KEY - file]$).
        return $MAC - KEY$, $app - KEY - file$ and $Signed - MAC[app - KEY - file]$.
    end if
end procedure
Activate the application using $app - KEY - file$.
Store and protect the values of $MAC - KEY$, and $Signed - MAC[app - KEY - file]$.
End

Whenever the application starts up: Every time the application is running, the newly proposed algorithm 2 ensures mobile authenticity, i.e., it checks whether the application is installed on a legitimate mobile.

Algorithm 2 Mobile Authenticity

Input: MPN, $IMEI$, $MAC{-}KEY$
Result: Ensure the mobile authenticity
Begin
Generate a new $app{-}KEY{-}file$ using MPN, $MAC{-}KEY$, and $IMEI$ of the current machine.
Construct a new $MAC[app{-}KEY{-}file]$.
Construct the old $MAC[app{-}KEY{-}file]$ from the signed $MAC[app{-}KEY{-}file]$.
if the Old $MAC[app{-}KEY{-}file]$==the new $MAC[app{-}KEY{-}file]$ **then**
    Run the mobile application.
Else
    Abort running the mobile application.
end if
End

Whenever the app starts up and the mobile connected to the Internet: To maximize the copyrights protection, the newly proposed algorithm 3 ensures the app authenticity whenever the app's starts up and the mobile is connected to the Internet.

Algorithm 3 Mobile Authenticity over the Internet

Input: MPN, $IMEI$, $MAC{-}KEY$
Result: Ensure the mobile authenticity
Begin
Generate a new $app{-}KEY{-}file$ using MPN, $MAC{-}KEY$, and $IMEI$ of the current machine.
Construct a new $MAC[app{-}KEY{-}file]$.
procedure CHECK THE MOBILE AUTHENTICITY AT THE SERVER(MPN, $IMEI$, $MAC[app{-}KEY{-}file]$)
if registered MPN==true & registered IMEI==true **then**
    Construct a $signed{-}MAC[app{-}KEY{-}file]$ using the received $MAC[app{-}KEY{-}file]$
    return the new $Signed{-}MAC[app{-}KEY{-}file]$
else
    Revoke the operation.
end if
end procedure
if the Old $Signed{-}MAC[app{-}KEY{-}file]$==the New $Signed{-}MAC[app{-}KEY{-}file]$ **then**
    Run the mobile application.
Else
    Abort running the mobile application.
end if
End

Whenever the mobile is lost, changed or destroyed: In this case, the legitimate user may request new activation parameters for the new mobile. So, after download, install, and open the mobile app on the new mobile, the newly proposed algorithm 4 validates the legitimate user. Meanwhile, this algorithm creates new activation parameters for this new mobile.

Algorithm 4 A New Installation of The Application on a New Mobile

Input: MPN, $IMEI$
Result: Complete a new installation of the mobile app on a new mobile for a legal user
Begin
Install the app on a new mobile
procedure GENERATE A NEW ACTIVATION PARAMETERS AT THE SERVER(MPN, $IMEI$)
    if registered MPN == true **then**
        Generate and send a ValidationMessage to the user.
        procedure VALIDATE ACTIVATION PARAMETERS' REQUEST AT THE CLIENT SIDE(ValidationMessage)
            if The user asked for a new activation == true **then**
                Generate a response message to the company's server with IMEI of the current mobile device.
                return the response message.
            Else
                Revoke the operation
            end if
        end procedure
        if isLegitimateUser == ture **then**
            Revoke the old license associated with this mobile number.
            Create a new $app{-}KEY$ for this specific MPN and the new $IMEI$.
            return MAC−KEY, app−KEY−file and Signed−MAC[app−KEY−file].
        else
            Abort the operation
        end if
    else
        Abort the operation
    end if
end procedure
Use $app{-}KEY{-}file$ to activate the application.
Store and protect the values of the $MAC{-}KEY$, and $Signed{-}MAC[app - KEY - file]$.
End

The advantages of using the proposed algorithms can be listed as follows:

1. As illustrated in Sections Fig. 1 and 3.3, the newly proposed algorithms are secure and efficient. For instance, an adversary cannot bypass the protection due to the use of a secure channel to deliver the $H(Nonce \parallel CompanyMasterKey \parallel MEID)$.
2. The code is open and free to be used. It may be adapted and/or enhanced.
3. The new algorithms provide digital rights management (DRM) functionality internally. So, there is no need to send the source code to a third party to add the DRM services.
4. The new algorithms are applicable to all mobile application types (Android, iOS, etc.) [5].

4.2 Development of an android secure mobile application

This section presents an Android secure mobile application demonstration that resists illegal distribution. This demonstration provides DRM functionality for a secure mobile app. It utilizes the new algorithms proposed in Section 4.1.

An android mobile application demonstration: Due to the embedded security functions based upon the newly presented algorithms, the developed Android application prevents its illegal distribution. This application can be illustrated as follows:

Installation: At the beginning, a user purchases and downloads the mobile application from a trusted App Store. Immediately, the installation process will begin.

Figure 2 illustrates the different stages of the installation process.

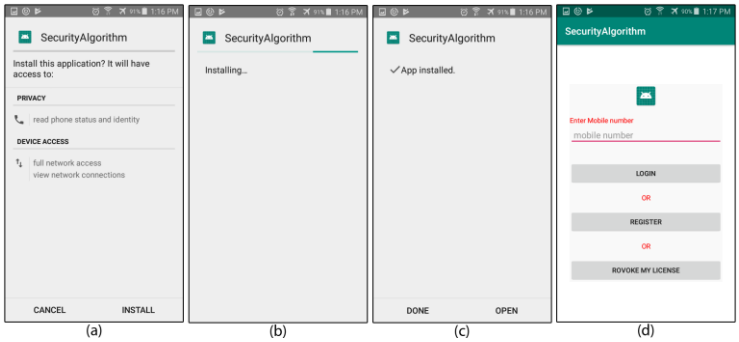

Fig. 2. Installation process: (a) get user permission. (b) installing the app. (c) Successful installation (d) the app's interface

Registration: According to algorithm 1, the user should register before login to the application. At the first opening of the application on a particular mobile, the user should prove his identity as a legal user who purchases the application. Figure 3 (a) shows that the user should register before login for the first time. In Figure 3 (b), the user enters his mobile number and then hits the register button. In Figure 3 (c), the application asks the user to enter the secret code. Meanwhile, the company's server stores the mobile data into a secure database and generates a random secret code for

this mobile number, as shown in Figure 3 (d). Then, the company's server sends this secret code to the user. Thereafter, the user enters the secret code as shown in Figure 3 (e). Once the user hits the submit button, the user logs in as shown in Figure 3 (f). The mobile should connect to the Internet during the registration process

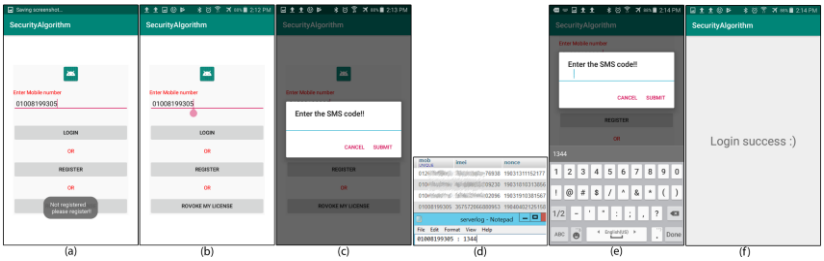

Fig. 3. Registration process: (a) login without registration. (b) begin the registration.(c) the secret code demands (d) the server generates the secret code (e) entering the secret code (f) successful login

Log In: When startup, the application should check the user's authenticity, as stated by the algorithms 2 and 3. A user cannot log in using an unregistered mobile number or an unregistered device as shown respectively in Figure 4 (a) and Figure 4 (b). To log in, the user should use a registered mobile number with a registered device as shown in Figure 4 (c) and Figure 4 (d).

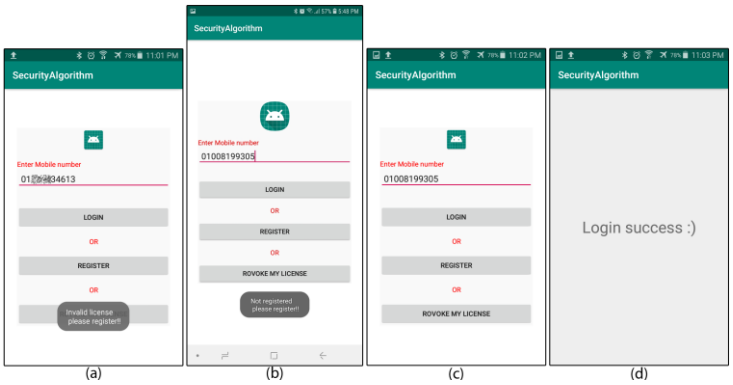

Fig. 4. Login process: (a) invalid MPN. (b) invalid device. (c) MPN and device are valid.(d) successful login

Acquiring a new licensing: A legitimate user may demand a new licensing whenever (s)he changed her/his mobile device. Firstly, a legitimate user downloads and installs the application as shown in Figure 5 & Figure 6 illustrates that the user cannot log in or register before (s)he revokes the old license and demands a new one. *Figure 6* (c) illustrates that the user cannot register without an Internet connection. To log in, the legitimate user should revoke the old license and demand a new one by pressing "REVOKE MY LICENSE" button, as shown in Figure 7 (a). Afterwards, the

company's server creates an activation code and sends it to the legitimate user via SMS, as shown in Figure 7 (b). Subsequently, the user enters the correct activation code and press' the "SUBMIT" button, as shown in Figure 7 (c). Figure 7 (d) shows that the new mobile is registered and log in into the application. The mobile should be connected to the Internet during the whole process.

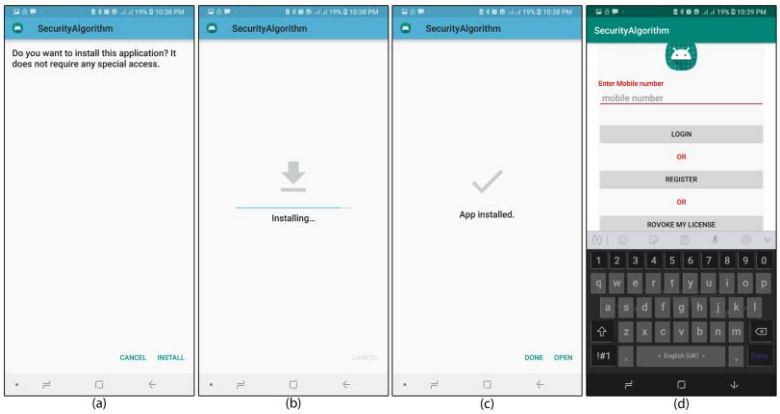

Fig. 5. Installation process on a new mobile: (a) get user permission. (b) installing the app.(c) successful installation (d) the app's interface

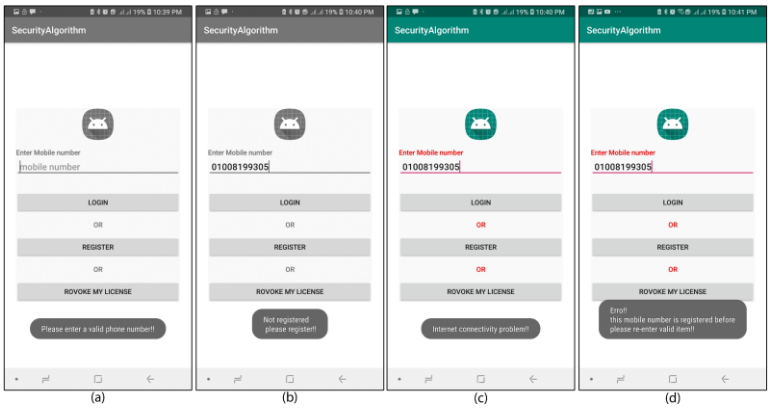

Fig. 6. Login or register from the new mobile before renewing the license: (a) login with empty MPN (b) login from invalid device (c) no Internet connection (d) register a new device before renewing the license

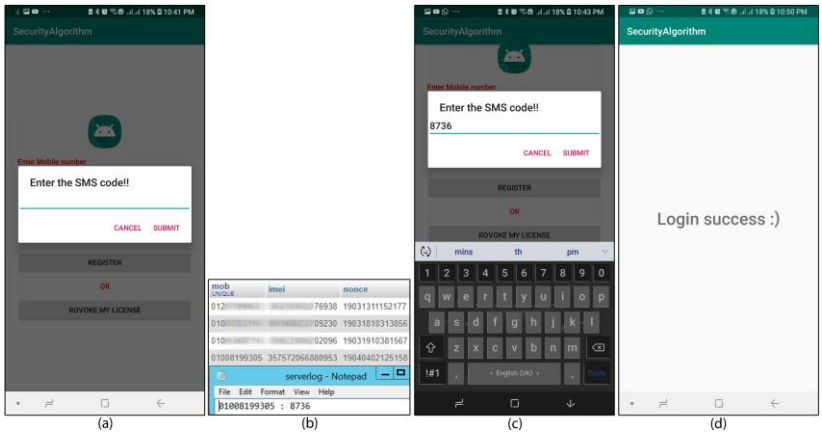

Fig. 7. Acquiring a new licensing: (a) the secret code demand. (b) the server generates the secret code. (c) entering the secret code. (d) successful login

The Case Study (Virtour mini-game): VirTour is a new paradigm of virtual tourism [31]. It is a gaming platform which introduces a new technology-based approach to site attraction in Egypt. As a mobile application, the VirTour's platform needs to be protected against all types of attacks. So, developers could use the newly presented architecture to protect VirTour's platform. Section 4.1 presented algorithms that protect mobile applications against illegal distribution. So, VirTour's developers may use those algorithms to protect their platform against illegal distribution as shown in Section 4.2.

5 Conclusion

Protecting software applications is a critical task for application developers. This paper aimed at providing an optimized security solution for developers. Accordingly, it surveyed the current security mechanisms for software applications. Then, it proposed a new software security optimization architecture (SoSOA). SoSOA may guide software developers to secure their software applications. Furthermore, this paper adapted the software security optimization architecture to suit mobile applications. Subsequently, it proposed security algorithms and its implementation for securing mobile applications (such as VirTour gaming platform) against illegal distribution.

The further work includes generating a new framework and security libraries which reflect the proposed software security optimization architecture and its adaptation for mobile applications.

6 Acknowledgement

This research is funded by the Academy of Scientific Research and Technology (ASRT), Cairo, Egypt, project titled "VirTour - New Paradigm of Virtual Tourism", (project ID: 1494).

7 References

[1] V. A. Padaryan and I. N. Ledovskikh, "On the Representation of Results of Binary Code Reverse Engineering," Programming and Computer Software, vol. 44, no. 3, pp. 200-206, 2018/05/01 2018, https://doi.org/10.1134/s0361768818030064

[2] D. Geethanjali, T. L. Ying, M. W. J. Chua, and V. Balachandran, "AEON: Android Encryption based Obfuscation," in Proceedings of the Eighth ACM Conference on Data and Application Security and Privacy, Tempe, AZ, USA, 2018: ACM, pp. 146-148, https://doi.org/10.1145/3176258.3176943

[3] N. Nabeel, M. H. Habaebi, N. A. Che Mustapha, and M. R. Islam, "IoT Light Weight (LWT) Crypto Functions," International Journal of Interactive Mobile Technologies (iJIM), Diffusion; lightweight hashing techniques; Mersenne number; energy efficiency. vol. 13, no. 04, p. 13, 2019-04-10 2019, https://doi.org/10.3991/ijim. v13i04.10524

[4] Y. Wang, D. Gong, B. Lu, F. Xiang, and F. Liu, "Exception Handling-Based Dynamic Software Watermarking," IEEE Access, vol. 6, pp. 8882-8889, 2018, https://doi.org/10.1109/access.2018.2810058

[5] K. Kumar and P. Kaur, "A Comparative Analysis of Static and Dynamic Java Bytecode Watermarking Algorithms," in Software Engineering, Singapore, M. N. Hoda, N. Chauhan, S. M. K. Quadri, and P. R. Srivastava, Eds., 2019// 2019: Springer Singapore, pp. 319-334, https://doi.org/10.1007/978-981-10-8848-3_31

[6] I. A. Aljazaery, H. T. S. Alrikabi, and M. R. Aziz, "Combination of Hiding and Encryption for Data Security," International Journal of Interactive Mobile Technologies (iJIM), embedding and encryption, exponential function, information security technique, Wavelet transformer. vol. 14, no. 09, p. 14, 2020-06-17 2020, https://doi.org/10.3991/ijim. v14i09.14173

[7] N. Yashenkova. "DRM for Android apps licensing and copy protection." StarForce Technologies Inc. http://www.star-force.com/blog/index.php?blog=2695 (accessed 22/11/2020, 2020).

[8] C. A. Ardagna, M. Cremonini, E. Damiani, S. De Capitani di Vimercati, and P. Samarati, "Location Privacy Protection Through Obfuscation-Based Techniques," in Data and Applications Security XXI, Berlin, Heidelberg, S. Barker and G.-J. Ahn, Eds., 2007// 2007: Springer Berlin Heidelberg, pp. 47-60 https://doi.org/10.1007/978-3-540-73538-0_4

[9] J.-M. Borello and L. Mé, "Code obfuscation techniques for metamorphic viruses," Journal in Computer Virology, vol. 4, no. 3, pp. 211-220, 2008/08/01 2008. https://doi.org/10.1007/s11416-008-0084-2

[10] I. You and K. Yim, "Malware Obfuscation Techniques: A Brief Survey," in 2010 International Conference on Broadband, Wireless Computing, Communication and Applications, 4-6 Nov. 2010 2010, pp. 297-300, https://doi.org/10.1109/bwcca. 2010.85

[11] A. Cimitile, F. Martinelli, F. Mercaldo, V. Nardone, and A. Santone, "Formal Methods Meet Mobile Code Obfuscation Identification of Code Reordering Technique," in 2017 IEEE 26th International Conference on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE), 21-23 June 2017 2017, pp. 263-268, https://doi.org/10.1109/wetice.2017.23

[12] M. D. Preda and F. Maggi, "Testing android malware detectors against code obfuscation: a systematization of knowledge and unified methodology," Journal of Computer Virology and Hacking Techniques, vol. 13, no. 3, pp. 209-232, 8 2017, https://doi.org/10.1007/s11416-016-0282-2

[13] Y. Zhou et al., "ProGuard: Detecting Malicious Accounts in Social-Network-Based Online Promotions," IEEE Access, vol. 5, pp. 1990-1999, 2017, https://doi.org/10.1109/access.2017.2654272

[14] I. Kinash and M. Dudarev. "DexProtector - Mobile Application Security." Licel Corporation. https://dexprotector.com/ (accessed 22/11/2020, 2020)
[15] [15] R. Caers and E. Lafortune. "Protecting Android applications and SDKs against reverse engineering and hacking." Guardsquare nv. https://www.guardsquare.com/en/products/dexguard (accessed 22/11/2020, 2020)
[16] W. Martin, F. Sarro, Y. Jia, Y. Zhang, and M. Harman, "A Survey of App Store Analysis for Software Engineering," IEEE Transactions on Software Engineering, vol. 43, no. 9, pp. 817-847, 2017, https://doi.org/10.1109/tse.2016.2630689
[17] D. A. Cooper, L. Feldman, and G. A. Witte, "Protecting Software Integrity Through Code Signing," 23/5/2018. [Online]. Available: https://www.nist.gov/publications/protecting-software-integrity-through-code-signing
[18] D. A. Almeida, G. C. Murphy, G. Wilson, and M. Hoye, "Investigating whether and how software developers understand open-source software licensing," Empirical Softw. Engg., vol. 24, no. 1, pp. 211–239, 2019, https://doi.org/10.1007/s10664-018-9614-9
[19] D. Schürmann, S. Dechand, and L. Wolf, "OpenKeychain: An Architecture for Cryptography with Smart Cards and NFC Rings on Android," Proc. ACM Interact. Mob. Wearable Ubiquitous Technol., vol. 1, no. 3, p. Article 99, 2017, https://doi.org/10.1145/3130964
[20] M. Hussain et al., "Conceptual framework for the security of mobile health applications on Android platform," Telematics and Informatics, vol. 35, no. 5, pp. 1335-1354, 2018/08/01/ 2018, https://doi.org/10.1016/j.tele.2018.03.005
[21] G. L. Masala, P. Ruiu, and E. Grosso, "Biometric Authentication and Data Security in Cloud Computing," in Computer and Network Security Essentials, K. Daimi Ed. Cham: Springer International Publishing, 2018, pp. 337-353.https://doi.org/10.1007/978-3-319-58424-9_19
[22] P. V L, "A Novel Authentication Mechanism to Prevent Unauthorized Service Access for Mobile Device in Distributed Network," International Journal of Interactive Mobile Technologies (iJIM), Authentication, Unauthorized Service Access, Mobile, Distributed Network vol. 12, no. 8, p. 16, 2018-12-24 2018,https://doi.org/10. 3991/ijim.v12i8.8194
[23] F. Cheng, Exploring Java 9: Build Modularized Applications in Java. Apress, 2018.
[24] M. Mabey, A. Doupé, Z. Zhao, and G.-J. Ahn, "Challenges, Opportunities and a Framework for Web Environment Forensics," in Advances in Digital Forensics XIV, Cham, G. Peterson and S. Shenoi, Eds., 2018// 2018: Springer International Publishing, pp. 11-33. https://doi.org/10.1007/978-3-319-99277-8_2
[25] J. Clarke-Salt, SQL Injection Attacks and Defense, 2nd ed. Syngress, 2012, p. 576.
[26] R. Elmasri and S. B. Navathe, Fundamentals of Database Systems, 7th ed. Pearson India, 2017.
[27] S. Agrawal and U. Singh, "Prevention of SQL Injection Attack in Web Application with Host Language," International Research Journal of Engineering and Technology (IRJET), vol. 4, no. 11, pp. 1468-1470, 2017. [Online]. Available: https://www.irjet.net/archives/V4/i11/IRJET-V4I11268.pdf
[28] Z. S. Alwan and M. F. Younis, "Detection and Prevention of SQL Injection Attack: A Survey," International Journal of Computer Science and Mobile Computing, vol. 6, no. 8, pp. 5-17, 2017.
[29] H. El-Sofany and S. Abou El-Seoud, "A Novel Model for Securing Mobile-based Systems against DDoS Attacks in Cloud Computing Environment," International Journal of Interactive Mobile Technologies (iJIM), mobile computing; cloud computing; mobile security; mobile attacks; denial of service attacks; distributed denial-of-service attacks vol. 13, no. 01, p. 14, 2019-01-29 2019, https://doi.org/10.3991/ijim. v13i01.9900
[30] M. McCamon and H. Blankenship, "OWASP Top 10 - 2017: The Ten Most Critical Web Application Security Risks." [Online]. Available: https://owasp.org/www-pdf-

archive/OWASP_Top_10-2017_%28en%29.pdf.pdf. https://doi.org/10.1007/978-3-642-16120-9_10

[31] H. Sayed. "VirTour." https://youtu.be/AD94Oa1wNi4 (accessed 22/11/2020, 2020).

8 Authors

Dr. Amr Abozeid is an assistant professor of computer science at Computer Science Department, College of Science & Arts (Gurayat), Jouf University, Saudi Arabia. He also worked as assistant professor of Computer Science at the Mathematics and Computer Science department, Faculty of Science, Al-Azhar University. His fields of research include video processing, computer vision, deep learning, and mobile computing.

Dr. AbdAllah A. AlHabshy is an assistant professor of computer science at Mathematics department, Faculty of Science, Al-Azhar University. His fields of research are Cryptography, Network Security, Mobile Security, Database Security, Software Security, Internet of things, and Video Protection. Email: AbdAllah@Azhar.edu.eg

Prof. Kamal Abdelraouf ElDahshan is a professor of Computer Science and Information Systems at Al-Azhar University in Cairo, Egypt. At Al-Azhar, he founded the Centre of Excellence in Information Technology, in collaboration with the Indian government, and was also the founder and former president of the coordination bureau of the Egyptian Knowledge Bank, the country's largest initiative for academic access. Among other accolades, he is a Fellow of the British Computing Society, and a Founding Member of the Egyptian Mathematical Society.

Article submitted 2020-11-29. Resubmitted 2021-03-27. Final acceptance 2021-03-27. Final version published as submitted by the authors.