A REUSABLE SOFTWARE COPY PROTECTION USING HASH RESULT AND ASYMETRICAL ENCRYPTION

Aswin Wibisurya; Timothy Yudi Adinugroho

Computer Science Department, School of Computer Science, Binus University
Jl. K.H. Syahdan No. 9, Palmerah, Jakarta Barat 11480
awibisurya@binus.edu, tadinugroho@binus.edu

ABSTRACT

Desktop application is one of the most popular types of application being used in computer due to the one time install simplicity and the quick accessibility from the moment the computer being turned on. Limitation of the copy and usage of desktop applications has long been an important issue to application providers. For security concerns, software copy protection is usually integrated with the application. However, developers seek to reuse the copy protection component of the software. This paper proposes an approach of reusable software copy protection which consists of a certificate validator on the client computer and a certificate generator on the server. The certificate validator integrity is protected using hashing result while all communications are encrypted using asymmetrical encryption to ensure the security of this approach.

Keywords: *software copy protection, hash, asymmetrical encryption*

ABSTRAK

Aplikasi desktop adalah salah satu jenis yang paling populer dari aplikasi yang digunakan dalam komputer karena satu kali install dan aksesibilitas cepat sejak komputer dihidupkan. Pembatasan jumlah instalasi serta legalitas pemakaian aplikasi desktop per pengguna sejak lama telah menjadi perhatian utama dari pihak pengembang aplikasi. Untuk alasan keamanan, perangkat lunak yang bertugas untuk melindungi aplikasi dari tindakan pemakaian ilegal, biasanya diintegrasikan dengan aplikasi tersebut. Hal ini tidak mendukung prinsip penggunaan kembali yang merupakan salah satu prinsip pengembangan aplikasi. Penulisan ini mengusulkan sebuah metode perlindungan perangkat lunak dari penduplikasian ilegal melalui pendekatan penggunaan kembali yang terdiri dari validator sertifikat di komputer klien, dan generator sertifikat di sisi server. Integritas dari validator sertifikat dijaga menggunakan hasil hash dan semua data yang dikomunikasikan dienkripsi menggunakan metode asimetrik untuk menjaga keamanan dari metode ini.

Kata kunci: *software copy protection, hash, asymmetrical encryption*

INTRODUCTION

Desktop application is one of the most popular types of application being used in computer due to the one time install simplicity and the quick accessibility from the moment the computer being turned on. However the one-time installation of the desktop application also raise problems regarding the user's rights and capabilities in copying, duplicating and even distributing the desktop applications software to other party without the application provider's consent. Software copy protection is a method designed to counter such problems. By planting extra file(s) or data such as license key to be used as the application legality validator, the software creator are able to provide some degree of countermeasure against user tendency to illegally duplicate and distribute the software. Users knowledgeable in computer programming might be able to decode information or duplicate a license file. Encryption method and hash result technique are common ways to ensure information hiding and file integrity.

Software copy protection reusability has also become a challenge on its own. To be able to reuse such complex mechanism also means leaner program, reduced development time and ultimately development cost which will affect the product's price. This paper focuses on a reusable software copy protection approach. A hashing technique and encryption method are used to ensure the integrity of reusable components as well as the certificate generated.

Literature Review

Reusable Software Component

Nowadays, there is a shift to reusing components in software development. According to AL-Badareen, *et al* (2011), the software reuse became a base of most software design because it can reduce cost, effort, time, and increase the quality and productivity of the software development process. Sojer and Henkel (2010) stated that efficiency and effectiveness are the two main purposes that a developer choose to reuse software code or components. Sojer and Henkel (2010) also stated that a software component to be reused needs to be a component with proper modification just as needed. Nobody will ever know what condition may appear in the future which drives modification of part of a software component. Therefore, a reusable software component needs to be as general as possible to avoid uneccessary future modification.

Software Copy Protection

Software industry uses software copy protection to prevent software usage by users who have not purchased the software yet. This method is used by companies who sell commercial software product. There are many software copy protection techniques. Bahaa-Eldin and Sobh (2014) stated that these techniques include these common points: (1) Software authentication – a copy protected software should check for the existence of a valid private certificate. (2) License integrity – the private certificate for authentication needs to proved that it is not altered. (3) Entity authentication – each license should be linked with a hardware/media to prevent license redistribution. (4) Protection security – the private certificate should be encrypted, so it cannot be altered or read by anyone.

Bahaa-Eldin and Sobh (2014) proposed an approach for software copy protection based on the four common points above. The technique requires authentication via a certificate generation over the internet. This certificate is encrypted using public-private key mechanism, so that the license integrity can be achieved. The software also checks for a USB dongle, so that every software copied into different computers can only be used with the existence of one USB dongle. The certificate contains a sequence number which is validated in the activation server through internet connection. Using this approach, the license can be validated for its originality, expired date, and so on. However, this approach requires the software copy protection module to be integrated with with the main software. Therefore it does not support reusability. Continuous internet usage for license validation in activation server also has its own disadvantage. The software will not be available for use when the user does not have an internet connection. Another problem for companies which might arise from this approach is the expensive investment on USB dongle for every copy of the software.

Encryption

Encryption is used to hide information from untrusted party. The encrypted information will show a long string of unreadable/nonsensical character. To understand the real information, it should be decrypted first by a key. There are currently two types of encryption: symmetrical and asymmetrical encryption. The symmetrical encryption is faster than asymmetric encryption. It also uses lesser memory (Agrawal & Mishra, 2012). The symmetrical approach uses a single key to encrypt and decrypt the information. When the information is transferred via network, the key has to be transferred as well. This causes the encrypted information to be vulnerable to untrusted party intercept. The asymmetrical approach is much more secure than the symmetrical approach because it use public-private key method where everyone knows what the public keys is and they are able to encrypt their message with it. However, for each public key, there are only several trusted parties that already hold the private key which match the said public key. This private key is the one that can be used to decrypt the message which was encrypted by the matching public key (Barukab, *et al*, 2012). Since the private key is not transferred through the network, it cannot be acquired by the untrusted party.

Symmetric Encryption

Symmetric encryption is useful for keeping the confidentiality of a message. As stated before, symmetrical encryption uses a single key to encrypt and decrypt information. Symmetric encryption can be generally viewed like figure (???). Suppose ‘A’ wants to send information ‘X’ to ‘C’. ‘A’ use a single key to encrypt ‘K’ the information and turn it into nonsensical message ‘Y’. Now, if ‘B’ manage to eavesdrop the message ‘Y’, B will not be able to extract any useful information since the information ‘X’ has been encrypted into nonsensical message ‘Y’ and need to be decrypted first using the same single key ‘K’ that was used to encrypt the information.

When the message ‘Y’ is received by the intended recipient which is C, then the message should be decrypted using the key ‘K’ which might have been agreed upon before hand by ‘A’ and ‘C’ or transferred through other secure means from ‘A’ to ‘C’. Of course this means that if the key ‘K’ falls into ‘B’, then B would be able to encrypt the message and tampered with the information. Using only symmetric encryption, there is no way to ensure that the message integrity has not been tampered or to ensure that the message sender is indeed ‘A’. However the real problem with symmetric encryption is how to transfer the key ‘K’ securely to the intended party since it will somewhat defeat the purpose of the key unless it was agreed upon beforehand (Paar & Pelzl, 2009).

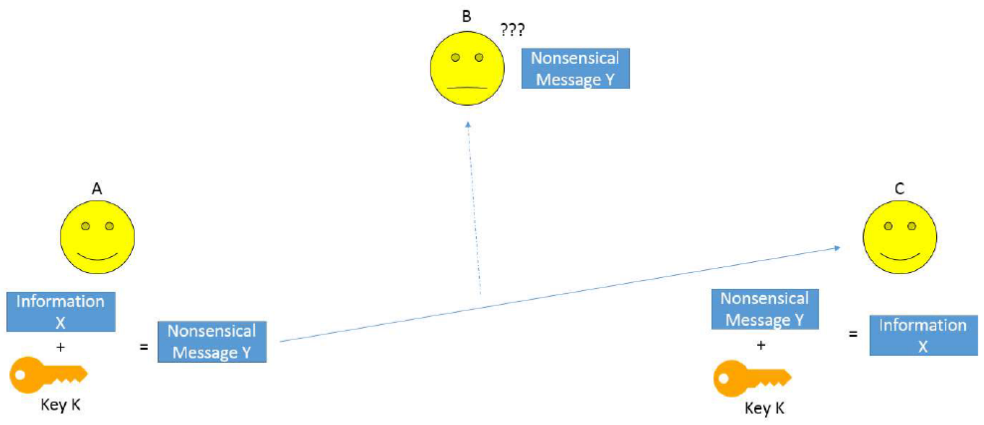

Figure 1 Symmetric Encryption

Asymmetric Encryption

Asymmetric encryption is developed to overcome the problem of key distribution in symmetric encryption. To put it simply, asymmetric encryption is like physical mailbox on the streets. Everyone is free to insert the mail, or encrypting the information using public key in this regards. However, everyone already has a way to open the mail securely using their own key or decrypting the message using theirown private key in this regards. Suppose ‘A’ wants to send information ‘X’ to ‘C’. ‘A’ then use C’s public key to encrypt the information and turn it into nonsensical message ‘Y’. Now, if ‘B’ manage to eavesdrop the message ‘Y’, B will only be able to decrypt the message using C’s private key. If C want to send information to A, then C will encrypt the information using A’s Public key and A will be able to decrypt the message using his own private key. This mechanism is quite safe since the private key is not shared or transferred. The only key that gets transferred is the public key which only useful to encrypt the information, not to decrypt it. This way, the message should be safe from unwanted party since they should not be able to get the private key which can be used to decrypt the message.

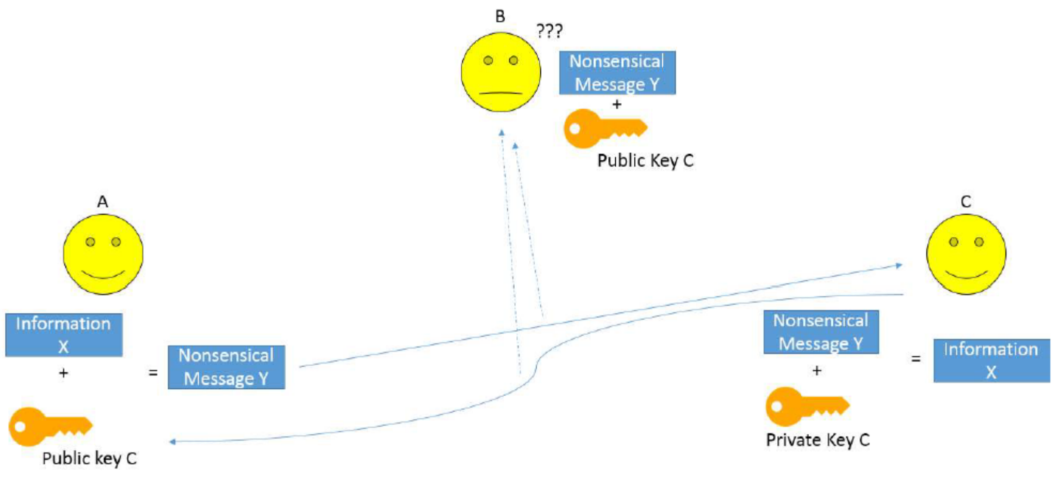

Figure 2 Asymmetric Encryption

File Integrity

Ensuring file integrity means ensuring that a file has not been modified by any person. A file which is stored on a client computer can be modified by the user himself or a malicious attack. An integrity check is performed to check whether a file has been modified from the last time or not. One of the common ways to perform integrity check is by using hash result (Sivathanu, Wright, & Zadok, 2005).

Hash Result

One of the popular mechanisms to perform file integrity check is by checking the file hash result. This mechanism is done by computing a hash using some popular hash function like MD5 or SHA1. The hash result should be stored in a manner which it cannot be easily modified. To check the file integrity, the hash result is once again computed and then compared to the previously stored hash result (Sivathanu, Wright, & Zadok, 2005). Hash result is very sensitive to change. Whenever an attribute or a portion of a file is modified, the hashing process will provide different hash result (Euresti *et al*, 2014). If the hash result is different, a file must have been modified by unauthorized person. Therefore, hash result can be used to ensure that a file is indeed the one from the creator and not modified by other party.

METHOD

The idea behind this research is that many kinds of software products need to be protected from copy and usage without permission. Because the steps for product registration and validation is usually the same and used by many products, the licensing process is a good candidate for reuse.

This paper’s approach consists of a certificate validator application and a certificate generator. The certificate validator needs to be deployed along with the software using the software copy protection so that the software can interact with the certificate validator. The certificate validator is responsible for two things: (1) Generating Activation Code which contains serial key and hardware machine code needed to create a new certificate. (2) Validating the machine code and expired date. The certificate generator resides on the server. The certificate generator accepts the Activation Code from the certificate validator, decrypts it, validates the serial key, saves the machine code – used for re-registration process – to the database and returns a generated certificate to be used.

As our approach deals with software copy protection, in which a software cannot work if it is copied without permission, there are several security issues that need to be handled. The second issue is the encryption of the Activation Code and the certificate. If somebody can figure out the encryption, a fake certificate could be created.

The first security issue is that the certificate validator might be replaced by a fake application behaving as if a valid certificate exists even though it doesn’t. To handle that issue, a md5 hash of the original certificate validator is stored on the application as a hardcoded field. Every time a software wants to interact with a certificate validator, the software needs to generate a md5 hash of the validator and compare the md5 hash to the one stored in its field. Because no other file can generate the same md5 hash, the replacement of the validator can be prevented.

To handle the second security issue, an asymmetrical encryption is implemented. The certificate validator holds public key A to encrypt activation code while the certificate generator holds private key A to decrypt the message. Meanwhile, the certificate generator holds public key B to encrypt certificate while the certificate validator hold private key B to decrypt the certificate before validating it. This way, no application can access the data except the particular target application. Also, without having the key, nobody can generate a certificate by his/her own.

The software copy can be registered using a given serial key for a certain amount of computers. The application will ask for the serial key if the certificate file cannot be found at a determined file path. The user can register the application for the first time, or in case the certificate file is lost, reregister the application using the serial key previously used for the same computer. The certificate generator on the server checks the existence of the machine code in the database. If the machine code doesn't exist, the software copy is assumed to be registered for the first time, the machine code is inserted to the database; hence decreases the amount left for another computers' registration. Meanwhile, if the machine code already exists, a certificate will be generated without decreasing the amount left for another computers' registration.

The database structure used in this approach is illustrated in the Entity Relationship Diagram (ERD) in figure 3.1. The *Applications* table contains various desktop applications which use this software copy protection. The *License* table contains serial key used for registration, its expiration date and the maximum quantity the software can be installed in different computers. The *Registered Computers* table stores the machine code for registered software copy on different computers.

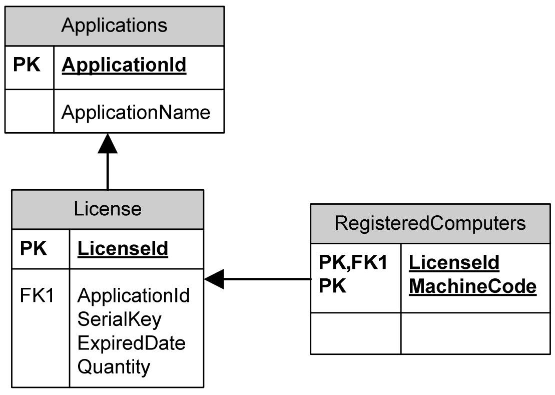

Figure 3 ERD of License Database

The detailed steps of our approach can be viewed in figure 4, which is explained as the following:

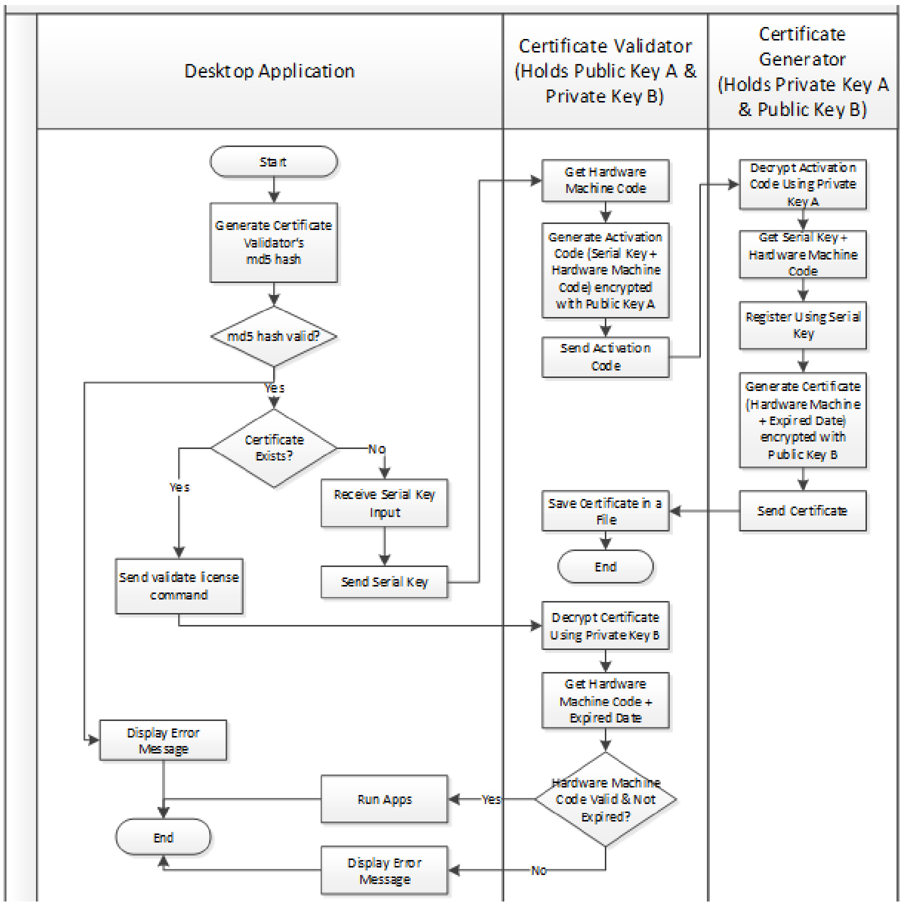
Desktop Application
Certificate Validator (Holds Public Key A & Private Key B)
Certificate Generator (Holds Private Key A & Public Key B)
Start
Generate Certificate Validator's md5 hash
md5 hash valid?
Yes
Certificate Exists?
No
Yes
Receive Serial Key Input
Send Serial Key
Send validate license command
Display Error Message
End
Run Apps
Display Error Message
Get Hardware Machine Code
Generate Activation Code (Serial Key + Hardware Machine Code) encrypted with Public Key A
Send Activation Code
Save Certificate in a File
End
Decrypt Certificate Using Private Key B
Get Hardware Machine Code + Expired Date
Hardware Machine Code Valid & Not Expired?
Yes
No
Decrypt Activation Code Using Private Key A
Get Serial Key + Hardware Machine Code
Register Using Serial Key
Generate Certificate (Hardware Machine + Expired Date) encrypted with Public Key B
Send Certificate

Figure 4 Steps of Certificate Generation and Validation

1. When the software starts, it generates md5 hash of the certificate validator and compare it with the original one.

2. If the md5 hash is invalid, the application displays error message and closes, else the application validates certificate.

3. The application searches for a certificate. If it is found, the application will request the certificate validator to validate it. The software copy is assumed as unregistered if the certificate doesn’t exist; hence the software will ask for a serial key. The serial key is usedfor registering the application for the first time or re-registering by using the same serial key as before.

4. In registration process, the application asks for a serial key, then send it to the certificate validator.

5. The certificate validator access the hardware machine code. The hardware machine code can be anything that is unique to a computer and is seldom changing like MAC address, CPU serial number, network card or combination of them.

6. The hardware machine code and serial key are encrypted into activation code using public key A and then sent to the certificate generator.

7. The certificate generator decrypts the activation code using private key A to get the hardware machine code and serial number.

8. The certificate generator registers the software identified by its serial key.

9. The hardware machine code and expired date are encrypted into certificate using the public key B. The certificate is sent to the certificate validator which saves the certificate as a file and validates it.

10. If the certificate exists, the certificate validator decrypts the certificate using private key B to get the hardware machine code and the expired date.

11. The certificate validator validates the hardware machine code and the expired date. The certificate validator will return a code specifying the validation result to the software.

12. According to the result, the software either shows an error message or starts the application.

RESULT AND DISCUSSION

This approach offers a one-time developed certificate validator and certificate generator to be used by limit the distribution of copies of different applications. Asymmetrical hash helps to ensure the certificate validator's integrity – that the certificate validator hasn't been replaced by another false program which only returns a valid value of certificate checking without actually checking the files.

This software copy protection's security is difficult to be compromised. Because the certificate is generated on the server side using the public key residing on the server, unless the user acquires access to the server and stole the public key, he/she will not be able to self-generate his/her own certificate.By storing the computer's machine code inside the certificate file, the software cannot be run when it is copied to a different computer as there is a unique machine code for every computer. Also, a *keygen* will not work because this approach requires to contact the server for the registration process.

This approach also has an advantage for computers that cannot always be connected to the internet. This approach only needs internet connection at the beginning of the registration process to communicate with the server for certificate generation. After the certificate has been generated, the certificate validator can work to validate the certificate without the need of internet connection. However, if the user of the application loses the certificate file, e.g. accidental deletion or new operating system installation, the software copy needs to be re-registered for a new certificate file. Re-registration is made possible by the storage of computer's machine code at the server. So, as long as the user still has the serial key and there is no changes on the computer's hardware, the machine code sent to the server will always be the same with the one stored on the server, thus enabling the re-registration process. Of course an internet connection is also needed for this re-registration process. This paper recommends the application to have an internet connection only at the first registration, each time the certificate is lost and each time the license expires.

In this approach, a periodic licensing is required. So the certificate file includes a pre-specified expired date. If a lifetime license is needed, the expired date part can be omitted from the certificate, so the certificate file consists of encrypted hardware code only. A prototype of the certificate validator and the certificate generator was built using Qt (C++) programming language. An experiment showed that the prototype can be used by several desktop applications developed using Qt (C++), C#.NET, and Java.

CONCLUSION

This paper proposed an approach to a reusable software copy protection which includes a certificate validator deployed with the software and a certificate generator on the server. As an interacting application, the certificate validator needs to be checked for its integrity. Md5 hash is chosen to ensure the application's integrity. To ensure that the certificate validator is not a self-generated certificate by the user, an asymmetrical encryption- public-private key – is used. The certificate is generated on the server, encrypted using a public key and can only be decrypted using the matching private key. By keeping the certificate validator on the client computer, the application will not need internet connection once the certificate has been created.

Some future study extensions from this paper might include the following: rather than determining one hardware machine code to use, this paper gives some options like MAC address, CPU serial number, network card, or combination of them. A research can be done to determine the best hardware machine code to use. A research can also be done to prove this approach's effectiveness to prevent a software from being copied and used without permission.

REFERENCES

Agrawal, M., & Mishra, P. (2012). A Comparative Survey on Symmetric Key Encryption Techniques. *International Journal on Computer Science and Engineering*, 877-882.

AL-Badareen, A. B., Selamat, M. H., Jabar, M. A., Din, J., & Turaev, S. (2011). Reusable Software Component Life Cycle. *International Journal of Computers*, 191-199.

Bahaa-Eldin, A. M., & Sobh, M. A. (2014). A Comprehensive Software Copy Protection and Digital Rights Management Platform. *Ain Shams Engineering Journal*.

Barukab, O. M., Khan, A. I., Shaik, M. S., & Murthy, M. R. (2012). Secure Communication using Symmetric and Asymmetric Crypthographic Techniques. *I.J. Information Engineering and Electronic Business*, 36-42.

Euresti, D., Smith, B., Chen, A., Sydell, A., Motes, A., Tang, J., & Hunter, R. (2014). *United States of America Patent No. US 20140122451A1.*

Paar, C., & Pelzl, J. (2009). *Understanding Cryptography.* New York: Springer.

Sivathanu, G., Wright, C. P., & Zadok, E. (2005). Ensuring data integrity in storage: techniques and applications. *StorageSS '05 Proceedings of the 2005 ACM Workshop on Storage security and survivability* (pp. 26-36). New York: ACM.

Sojer, M., & Henkel, J. (2010). Code Reuse in Open Source Software Development: Quantitative Evidence, Drivers, and Impediments. *Journal of the Association for Information Systems*, 868-901.