Scenarios for Securing Content Delivery in the DRM Environment

Antonius Cahya Prihandoko[1], Hossein Ghodosi[2], and Bruce Litow[2]

[1] University of Jember, East Java 68121, Indonesia
[2] James Cook University, Townsville, QLD 4811, Australia
antoniuscp.ilkom@unej.ac.id
{hossein.ghodosi/bruce.litow}@jcu.edu.au

Abstract. *In the DRM environment, content is usually distributed in an encrypted form. Typically, a secure encryption algorithm is utilized to accomplish such protection. However, executing this algorithm in an insecure environment may allow adversaries to compromise the system and obtain information about the decryption key. Keeping such a key secret is a major challenge for content distribution systems. We consider two solutions for securing content delivery. The first solution involves modifying the algorithm in such a way as to make implementation unintelligible. The second solution involves setting a buyer-seller protocol to communicate the key securely. In addition, the protocol can be set to achieve security for the content provider and privacy protection for user. This paper describes a study of these scenarios for DRM applications w.r.t securing content delivery.*

Keywords: *Digital Rights Management, Code Obfuscation, White-box Cryptography, One-time Program, Oblivious Transfer.*

1 Introduction

Secure content delivery is urgently required in digital content distribution systems to guarantee that only authorized users can access content. In addition, users should be able to access content privately, but no more than they are entitled to access. To achieve these goals, initially, content is distributed in an encrypted form. A secure encryption algorithm is typically used to accomplish the protection. However, in the Digital Rights Management (DRM) context, such an algorithm may be executed in an insecure environment. This situation may allow adversaries to compromise the system and obtain information about the decryption key.

Keeping the key from being accessible to users is a major challenge for the DRM application in a content distribution system. The protection scenario must be set to prevent the extraction of the information of the decryption key, even when the application is executed in an insecure environment. Indeed, the security of an application must merely rely on the security of the key (Kerckhoffs's principle). Keeping the key secret must be considered when securing content distribution. To achieve secure content de-livery, we consider two approaches: modifying cryptographic algorithm implementation and setting buyer-seller protocols.

Modifying the implementation of the encryption algorithm is intended to make the implementation unintelligible. Any adversary will be unable to reverse engineer or com-promise the system. Ultimately, this modification will save the key. Code obfuscation and white-box cryptography (WBC) are two common methods for such a modification. These methods are primarily aimed protecting software implementations. In this paper, we analyse the feasibility of code obfuscation and WBC in the DRM applications, and propose a protocol for the white-box implementation.

Another solution for securing content delivery is constructing a secure buyer-seller protocol. This protocol allows the provider to deliver content in such a way that the authorized users can privately access it according to their rights. For example, a customer who pays for a certain number of views of a movie cannot access the movie more than he is entitled to do. In this case, the decryption key allocated to the customer can only be used for a limited number of executions. This mechanism will ultimately achieve security for the

content provider and privacy protection for the users. In this paper we also propose a protocol to fulfill this need, based on the concept of one-time program and oblivious transfer.

2 Code Obfuscation and Its Feasibility

Code obfuscation is intended to protect software implementation. In this method, the program used to implement the algorithm is rewritten in such a way that certain characteristics of the original program are hidden and unintelligible. This mechanism is expected to prevent the program from being reverse engineered.

Theoretically, the definition of obfuscation was initiated by Barak et al. [1]. Ac-cording to the definition, a probabilistic algorithm O is an obfuscator if it satisfies:

- Functionality. For any program P , O(P) is a program that computes the same function as P .
- Virtual Black Box Property (VBBP). Anything that can be efficiently computed from O(P), can be efficiently computed given oracle access to P .

Obfuscation, however, is hard to achieve. Barak et al. [1] showed that there exists some predicates $\pi : F \rightarrow \{0, 1\}$ that can be computed efficiently when having access to an obfuscated implementation O(f) of $f \in F$, but no efficient algorithm can compute $\pi(f)$ much better than by random guessing, given oracle access to f. As a result, a generic obfuscator, i.e. an obfuscator that protects any given program, does not exist.

Both positive and negative results on code obfuscation were investigated. Some positive results of code obfuscation were applied and proved on point functions [13, 3, 4, 2]. However, another investigation showed the impossibility of such obfuscation [9]. In addition, a best-possible obfuscation can be achieved by relaxing the white-box requirement [9]. Other positive results of code obfuscation were also applied to cryptographic primitives, allowing secret-key cryptography to be converted into public-key cryptography [10], and are applicable to re-encryption function [11]. While general purpose obfuscators are currently impossible, obfuscators for simple functions may exist; and more positive results of program obfuscation are applied for cryptographic primitives and point functions.

In practice, obfuscation of a program is applied to the variables used in the program. Variable names are scrambled and data that were stored in a single variable are split into multiple variables and recombined at the execution time. This mechanism makes a code difficult for humans to understand, and thus effective for hiding an algorithm and protecting the code, but not for any encryption key used by the code. Additionally, code obfuscation is often integrated with code flattening. In code flattening, extra paths are introduced in the program structure. This technique makes the program difficult to analyse. However, it can be reverse engineered [15] and, thus, the main goal of code obfuscation cannot be achieved. Because of the fact of the practical obfuscation and the result of the theoretical aspect, code obfuscation is less applicable in the DRM implementation.

3 White Box Cryptography

The threat models for traditional cryptographic applications are the black-box attack models. These models assume that an attacker has no physical access to the system and can only control the input and output of the algorithm. With this assumption, the algorithm needs to be executed in a secure environment. However, this deployment is impractical and therefore not appropriate for DRM implementation. A more appropriate attack model for the algorithm used in DRM implementation is the white-box attack model. In this model, the attackers have full control over the whole operation and can freely observe dynamic code execution. Internal algorithm details are absolutely visible and alterable.

White-box cryptography (WBC) is an obfuscation technique intended to implement cryptographic primitives in a white-box attack model. Despite providing a fully trans-parent methodology, white-box cryptography integrates the cipher in such a way that does not reveal the secret key.

3.1 White-box Implementation

The basic notion of white-box implementations is to rewrite a key so that all information related to the key is hidden. External encoding can be used so that the encryption and decryption software requires encoded inputs, and produces encoded outputs. This encoding mechanism can be done by replacing the encryption function E_k with the composition $E_k^0 = G \circ E_k \circ F^{-1}$. The input encoding function F and the output decoding function G^{-1} must not be stored in the same platform that computes E_k^0, so that the white-box implementation cannot be used to compute E_k. This means that encoding input and decoding output have to be kept secret. At present, white box implementation cannot stand alone; it should be used in conjunction with other techniques to provide protection against key recovery attacks [12]. Although this scenario is not standard, such an approach is useful for many DRM implementations.

The first introduced white-box implementations were applied to the data encryption standard (DES) and advance encryption standard (AES) [5, 6]. These implementations provide secure protection of a secret key in a cryptographic module, but have a slow performance. For example, the memory size and operation of standard AES require 4352 bytes and 300 lookups, respectively, while white-box AES needs 770048 bytes and 3104 lookups, respectively [6]. Furthermore, to encrypt 1 MB data, standard AES takes less than 0.5 seconds, while white-box AES takes more than 3 seconds [18].

WBC has been deployed as a protection technique in DRM implementations. This technique is even suggested to be the most effective protection in DRM applications [15]. In practice, WBC combines the data transformation mechanism and lookup table, which allows decrypting content without directly revealing the key. The entire white-box algorithm is running inside a transformed boundary.

Although many cryptanalysis techniques have been published, so far in real-world products, no white-box implementation has suffered from a key extraction attack. In-deed, breaking white-box implementations in practice is hard and time consuming [17]. Despite the robustness of practical white-box implementations, performance and security are still the main concerns for current applications. Low performance and high-consumed memory size limit the application of WBC, especially for mobile devices. Although no attack on commercial white-box implementations has been found, successful attacks may be possible in the future.

3.2 A Proposed Protocol for White-box Implementation

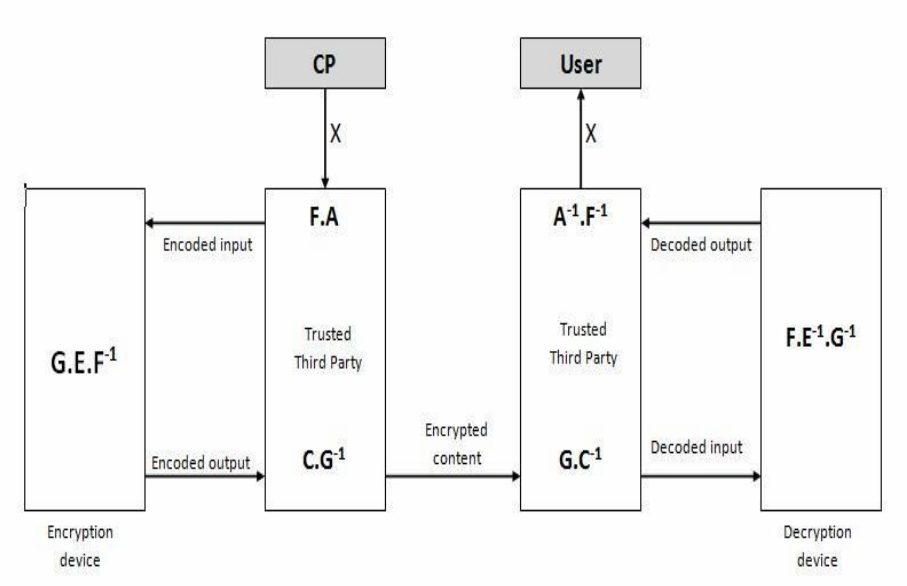

Fig. 1. Proposed WBC protocol

We propose a protocol for white-box implementation in a DRM application. White-box implementation cannot be stand alone. It needs external computations that have to be kept secret, i.e. act as a black-box. We suggest employing a trusted third party (TTP) in our proposed protocol to accomplish such computations. The TTP consists of two synchronized devices; each is placed in the content provider's encryption device and the user's decryption device. We assume that the TTP devices act as a black-box so that all computations in them are unaccessible. The scheme of the protocol is illustrated in Fig. 1.

Suppose content provider (CP) wishes to encrypt content (X) using a functions composition $C \circ E \circ A$. The implementation of this encryption can be obfuscated using the following mechanism.

1. Upon receiving content X, TTP generates input encoding function F and output encoding function G, and computes their inverses, F^{-1} and G^{-1}, respectively. TTP then computes $F \circ A(X)$ and sends it into the encryption device.
2. The encryption device encrypts $F \circ A(X)$ using function $G \circ E \circ F^{-1}$ and sends the result to the TTP.
3. TTP decodes the encryption result using function $C \circ G^{-1}$ yielding the encrypted content X^0 and sends it to the user's device.

The decryption process on the user side is performed in the following steps.

1. Upon receiving X^0, TTP decodes it using function $G \circ C^{-1}$ and sends the result into the decryption device.
2. The decryption device decrypts $G \circ C^{-1}(X^0)$ using function $F \circ E^{-1} \circ G^{-1}$ and sends the output back into the TTP.
3. TTP decodes the decryption output using function $A^{-1} \circ F^{-1}$ and yields the content X

In this scenario, the user can only access the decryption function $F \circ E^{-1} \circ G^{-1}$. Knowing only this function, the user cannot access the content. The original encryption function is absolutely hidden inside the TTP computations, which are unaccessible. The protocol is secure since some fundamental computations are performed inside "black-box" devices.

4 One-Time Programs

The one-time program (OTP), introduced by Goldwasser et al. [8], is a new computational paradigm for supporting security applications. An OTP can be executed once on a single input and then self destructs. Such a program can be extended to a k-time program which can be evaluated for k times and then self destruct. The program acts as a black-box as there is nothing about the program can be leaked except the computation result. To achieve this functionality, however, an OTP cannot be solely software based, as any software, in principle, can be copied and run more than once. One-time functionality can be achieved by means of secure hardware devices, namely one-time memory (OTM).

How does it work? OTP is a software-hardware package that combines one or more hardware devices H_1, ..., H_m and a standard software, i.e. a non-uniform Turing machine M. Note that M can be read and modified, whereas the devices can be accessed only via their input output (I/O) interfaces. An execution of OTP $P = M^{H1,...,Hm}$ on input $x \in \{0, 1\}^n$, is a run of $M^{H1,...,Hm}(x)$.

An OTM is a memory device which is initialed with two keys (k_0, k_1) and a tamper-proof bit (TPB) set to 0. Upon receiving a single bit input $b \in \{0, 1\}$, OTM verifies whether TPB = 0. If so, OTM sets TPB to 1 and outputs k_b, otherwise outputs an error symbol. OTM outputs one of its initial keys and the other key is irretrievably lost. The security assumptions from the OTM device include:

– memory locations that are not accessed by the device will not leak via a side channel, whereas a memory cell that is accessed may be immediately leaked.
– the single bit b is tamper-proof, but is readable.

The key point of the OTP is the OTM, a memory device that enables the OTP to be executed a limited number of times. By converting a sold program into an OTP, the vendor can put the restrictions on how many times the program can be used. The program is guaranteed not to be reverse-engineered and the components that restrict the number of executions cannot be removed [8]. This characteristic ensures that the OTP mechanism deals effectiely with copy protection and software protection problems.

The primary weakness of the Goldwasser et al.'s construction, however, is the size of the OTP [8]. In the

worst case, the ”corrupted circuit” part of the program (the software part) is as large as the running time of the original program. The size problem of the construction is likely to affect the cost of its implementation. Therefore, an open problem remains: whether it is possible to construct a small OTP with simple hardware that does very little work.

5 Oblivious Transfer

Oblivious Transfer (OT) is a cryptographic protocol that allows two parties to exchange one or more secret messages, where at the end of the protocol the sender does not know whether the recipient actually received the information. The first OT protocol, introduced by Michael Rabin [14], was intended to overcome the exchange of secrets (EOS) problem. In this protocol, the receiver can access the message with probability $\frac{1}{2}$ and the sender will not know whether the receiver can access the message.

The notion of Rabin's EOS protocol can be adopted for secret exchange between content providers and users in a digital transaction. The content provider sends the user the secret key that can be used to decrypt the protected content. In exchange, the user sends the content provider the secret key that can be used to access the user's personal information. However, in the context of a content distribution system, the protocol needs to be modified, as in such a system the probability of an authorized user to received content must be 1.

5.1 A Proposed OT Protocol for DRM Implementation

An oblivious transfer protocol has to be set up in such a way that it will achieve security for the sender and privacy for the receiver [7]. The former means that the receiver will not be able to learn more than he was supposed to learn. The latter means that the sender will not know what the receiver has learned.

We propose an oblivious transfer ”buyer-seller” protocol that is more flexible and appropriate for the DRM implementation. With only one assumption and an efficient computation, the protocol achieves unconditional security for the sender and privacy for the receiver. In addition, it may also provide a solution to the cost problem of the OTP construction.

Our protocol utilizes tamper-proof devices, such as smart-cards. A tamper-proof device means any device that can be used only in a particular way; otherwise the device will be corrupted and its content will not be accessible any more. Utilizing tamper-proof devices in this protocol is less expensive. The device contains only two types of functions and performs efficient computations. With this characteristic, the device can be mass produced at a low cost.

In this protocol, each device contains a pair of functions (GetKey, GetContent). GetKey allows the user to ask for the key; that is, the input parameter to the GetContent function. GetContent, on the other hand, requires an authorized key to reveal the message stored in it. Creating a single device containing all pairs of functions (GetKey, GetContent) may be reasonable and more efficient. However, for the purpose of a clear explanation in this paper, we assume that one device contains a pair of functions (GetKey, GetContent).

Suppose the content provider (Alice) provides N contents (e.g. movies), M_1, ..., M_N , and the customer (Bob) wishes to access K, where $K < N$, of these contents. Alice has a secret code S to access the contents, and utilizes Shamir's secret sharing scheme [16], with the threshold parameter $N - K$, to share the secret. That is, she split the secret into N pieces such that any set of at least $N - K$ shares can reconstruct the secret.

The scenario can also be applied when a customer requests N accesses, instead of N items. For example, a customer wants to watch a movie for K times. The movie provider then sends the customer a package containing N pairs (GetKey, GetContent) with the sharing threshold is $N - K$. In this case, all GetContent functions contains movie of the same title.

5.2 Analysis of Security and Privacy

Assuming that tamper-proof devices exist, the proposed protocol achieves unconditional security for the seller. In the proposed protocol, the contents are stored in tamper-proof devices. The buyer cannot access content without obtaining the secret key. The key, however, is split into several pieces of shares and

distributed among the devices using Shamir's secret sharing scheme [16]. This scheme is secure because knowing less than a predetermined number of shares give the buyer no way to reconstruct the secret. Therefore, the buyer can only obtain the secret key if (and only if) he sacrifices all contents that he is not supposed to access. This means that the buyer is not able to access anything other than the contents that are supposed to be accessed.

With the same assumption, we claim that the proposed protocol preserves uncondi-tional privacy for the buyer. In the proposed protocol, there is no interaction between seller and buyer after the seller gives all devices to the buyer. There is no way for the seller to determine which devices the buyer has used. As all devices are corrupted at the end of the protocol, the seller has no knowledge about which content that has been accessed by the buyer. Therefore, the buyer's privacy is fully protected.

6 Concluding Remarks

Securing content delivery in DRM implementation relies on securing the decryption key. Some solutions attempt to modify the implementation of the encryption algorithm so that it will be unintelligible. We have observed code obfuscation and white-box cryptography in accomplishing such solutions and analysed their feasibility in the DRM implementations.

Constructing a secure buyer-seller protocol is another attempt to protect the de-cryption key. The protocol is intended to communicate the secret key such that the key can be used no more than it was supposed to be used. The notion of oblivious transfer and one-time program can be adopted to construct buyer-seller protocols for particular objectives. We proposed an oblivious transfer protocol which unconditionally achieves security and privacy with an e*ff*icient computation. With this characteristic, our protocol can be implemented at a low cost.

Most of the solutions for securing the decryption key rely on the role of an extra device which acts as a black-box. This means that all processes performed in this device cannot be observed, except the input and output. The information about the key is absolutely hidden inside computations in such a device.

References

1. Barak, B., Goldreich, O., Impagliazzo, R., Rudich, S., Sahai, A., Vadhan, S., Yang, K.: On the (im)possibility of obfuscating program. In: Kilian, J. (ed.) Advance in Cryptology - CRYPTO 2001: 21st Annual International Cryptology Conference. vol. LNCS 2139, pp. 1–18. Springer, Berlin (August 2001)
2. Bitansky, N., Canetti, R.: On strong simulation and composable point obfuscation. In: Rabin, T. (ed.) CRYPTO 2010. vol. LNCS 6223, pp. 520–537. Springer (2010)
3. Canetti, R., Dakdouk, R.R.: Obfuscating point functions with multibit output. In: Smart, N. (ed.) EUROCRYPT 2008. vol. LNCS 4965, pp. 489–508. Springer (2008)
4. Canetti, R., Rothblum, G.N., Varia, M.: Obfuscation of hyperplane membership. In: Micciancio, D. (ed.) TCC 2010. vol. LNCS 5978, pp. 72–89. Springer (2010)
5. Chow, S., Eisen, P., Johnson, H., van Oorschot, P.C.: A white-box DES implementation for DRM applica-tions. In: Feigenbaum, J. (ed.) DRM 2002. vol. LNCS 2696, pp. 1–15. Springer-Verlag, Berlin Heidelberg (2003)
6. Chow, S., Eisen, P., Johnson, H., van Oorschot, P.C.: White-box cryptography and an AES implementation. In: Nyberg, K., Heys, H. (eds.) SAC 2002. vol. LNCS 2595, pp. 250–270. Springer-Verlag, Berlin Heidelberg (2003)
7. Ghodosi, H.: A general model for oblivious transfer. In: Proceedings of the Sixth International Workshop for Applied PKC. vol. IWAP2007, pp. 79–87. Perth, Australia (December 2007)
8. Goldwasser, S., Kalai, Y.T., Rothblum, G.N.: One-time programs. In: Wagner, D. (ed.) CRYPTO 2008. vol. LNCS 5157, pp. 39–56. International Association for Cryptologic Research 2008 (2008)
9. Goldwasser, S., Rothblum, G.N.: On best-possible obfuscation. In: Vadhan, S.P. (ed.) TCC 2007. vol. LNCS 4392, pp. 194–213. Springer, Amsterdam, The Netherland (2007)
10. Hofheinz, D., Malone-Lee, J., Stam, M.: Obfuscation for cryptographic purposes. In: Vadhan, S.P. (ed.)

TCC 2007. vol. LNCS 4392, pp. 214–232. Springer, Amsterdam, The Netherland (2007)
11. Hohenberger, S., Rothblum, G.N., Shelat, A., Vaikuntanathan, V.: Securely obfuscating re-encryption. In: Vadhan, S.P. (ed.) TCC 2007. vol. LNCS 4392, pp. 233–252. Springer, Amsterdam, The Netherland (2007)
12. Joye, M.: On white-box cryptography. In: Elci, A., Ors, S., Prencel, B. (eds.) Security of Information and Networks. pp. 7–12. Traff ord Publishing (2008)
13. Lynn, B., Prabhakaran, M., Sahai, A.: Positive results and techniques for obfuscation. In: Cachin, C., Camenisch, J. (eds.) Advances in Cryptology - EUROCRYPT 2004. vol. LNCS 3027, pp. 20–39. Springer (2004)
14. Rabin, M.O.: How to exchange secrets with oblivious transfer. Technical Report TR-81, Aiken Computation Lab, Harvard University (1981)
15. Schultz, R.: The many facades of DRM. MISC HS 5 Magazine pp. 58–64 (April 2012)
16. Shamir, A.: How to share a secret. Communications of the ACM 22(11), 612–613 (1979)
17. Wyseur, B.: White-box cryptography: Hiding keys in software. MISC HS 5 Magazine pp. 65–72 (April 2012)
18. Yoo, J., Jeong, H., Won, D.: A method for secure and efficient block cipher using white-box cryptography. In: 6th International Conference on Ubiquitos Information Management and Communication. vol. ICUIMC'12 (2012)