A Flexible and Open DRM Framework

Kristof Verslype and Bart De Decker

Department of Computer Science, K.U.Leuven,
Celestijnenlaan 200A, B-3001 Leuven, Belgium
{kristof.verslype,bart.dedecker}@cs.kuleuven.be

Abstract. Current DRM implementations rely on obfuscating the inner working of the DRM client. Moreover, the rights to consume content are rather device bound than person bound. We present a first step towards an open DRM framework which is based on the security of its building blocks. The presented framework binds the right to consume content to persons instead of to devices. An extension of the current TPM specification is proposed to enhance the security of DRM clients.

Keywords: DRM, Digital Rights Management, TPM, smart card, e-commerce

1 Introduction

DRM (Digital Rights Managent) is a technique to allow owners of digital content to control access to and distribution of this content and to restrict its usage in various ways which can be specified by the owner or his/her delegates.

Current implementations are closed source and most details are hidden, because, currently, the security of the DRM technologies relies on the secrecy of algorithms in the DRM client. Due to this approach, even the DRM technologies that are considered as the most mature and most secure are broken (see [7]).

On the other hand, current technologies lack flexibility in many ways. This paper is intended to be a next step towards more flexible DRM. Firstly, the ability to consume content is bound to the consumers themselves, where currently, this is bound to one or more devices. Secondly, the presented framework is flexible in the sense that it is based on building blocks that can be replaced if they are no longer considered appropriate. The required building blocks are not all equally mature, but we can expect that this will impove in the near future.

The basis of the person binding solution is to have two types of licenses: a content license and a root license. A content license gives a specific consumer the right to perform some actions on DRM protected content. A root license is device bound and enables the consumer to use all his content licenses on that device. A reasonable extension of the current TPM specification is proposed in order to obtain a DRM framework that is hard to break.

In the next section, the general DRM concepts are introduced. In section 3, some building blocks are briefly explained. Section 4 presents the DRM frame-

work. Section 5 analyses the security properties. Section 6 compares the construction with existing implementations. The paper ends with the conclusions and future work in section 7.

2 General DRM Concepts

This section introduces the main concepts of DRM. An introduction to the technical aspects of DRM can be found in [6].

Content refers to the data to protect. This can be multimedia, text, applications or other data constructs. Performing an action on content is called content *consumption*. The *producer* is the entity that owns the rights to distribute and sell content. The *consumer* obtains and consumes content and the *publisher* owns and manages the DRM system used to distribute content. The online *DRM system* is the (set of) server(s) offering DRM services to consumers and producers. The *DRM client* is the entity at the consumer side that is responsible for securely performing the DRM-specific operations such as the enforcement of the usage rules (the rights).

Usually, a *prevention mechanism* (encryption) is combined with a *detection mechanism*, which allows for identifying the source of misuse when illegally distributed content is found. Usage rules are associated with the corresponding DRM protected content. *Licenses* introduce a separation of DRM protected content and the usage rules associated with it. The latter describe the actions allowed on the content. Before consumers are able to use protected content, they first have to obtain a corresponding license that enables them to use the content according to the usage rules described therein.

A *contract* can be agreed between publisher and producer stating the terms of the agreement (e.g. royalties and usage rules of the licenses).

3 Cryptographic Primitives and Techniques

Besides the classical cryptographic primitives such as hash functions, digital signatures, certificates, public and private encryption, some less commonly known techniques will be needed to develop the DRM framework.

Watermarking (see [2]) embeds some information into content without noticeably changing the content. The watermark has to be undetectable by human perception. Strong watermarks survive manipulations such as D/A A/D conversions. It is one of the least mature technologies used in DRM.

White-box cryptography (see [3], [4]) assumes that the adversary can fully analyse the software implementation and the run-time instances, including the execution of cryptographic functions. White-box cryptography embeds a key in code such that it remains hidden from adversaries.

Code guarding (see [5], [1]) is a collection of techniques used to prevent tampering with the code during execution. Code guarding is useful when applications run on untrusted hosts.

TPM (Trusted Platform Module, see [12]) was specified by the Trusted Computing Group. The TPM guards the system security status during startup and

runtime. This status can also be interrogated later on. The TPM has some private keys embedded that cannot be read by anyone. These all have their corresponding public keys and certificates. The TPM offers the possibility to protect key material for outsiders, to authenticate the system to third parties, to prove the system's security status to third parties, to generate random numbers, to seal content and to detect configuration changes. On top of the TPM, a Trusted environment is built and an API is offered to the applications.

The TPM has its own cryptographic co-processor which cannot be addressed directly from outside the TPM. The TPM has volatile and non-volatile memory which can be used to store information in the TPM. Authentication to the TPM is based on knowledge of a shared secret. Other entities can get authorized by the TPM owner to access the TPM. It is possible to establish a confidentiality protecting transport session between (remote or local) processes and the TPM with the consent of the TPM owner. The TPM v1.2 has its own mechanism to do access control of software processes to the TPM. The TPM has a symmetric encryption engine to protect the confidentiality in transport sessions, to encrypt data that is stored outside the TPM, ... The TPM can store data such as keys in an encrypted form on the hard disk, such that it can only be decrypted by that TPM. The data is also integrity protected. The TPM v1.2 specifications offers monotonic counters. A process can get exclusive access to a monotonic counter such that it can be read out and incremented by that process. The TPM's Tick Counter can, with some external support enable secure time stamping. The TPM specification describes more functionality, but these are the most crucial ones for the DRM framework.

4 Framework

In this section, the framework is elaborated. We first divide the online DRM system into components. We then give a high level overview of our approach, followed by the assumptions made. Finally, the framework itself is described.

4.1 Components

In [8] different components in a generic DRM architecture were identified. We slightly adapt these to our specific needs. The result is shown in figure 1. In the center, we have the online DRM System, which consists of multiple services that are available to the consumer (via the DRM client) or the producer (via a producer tool). The *Import Service (IS)* is used to add new content to the online DRM system. The *Content Service (CS)* and *License Service (LS)* provide content and corresponding licenses. The *Registration Service (RS)* enables or disables a device to consume the consumer's content. The *Identification Service (IdS)* identifies the source in case illegally distributed content is found. The Client Setup Service is contacted to install or update a DRM client. The black arrows indicate communication initiated by (and with the consent of) the consumer or producer. The grey arrows indicate communication between the online services.

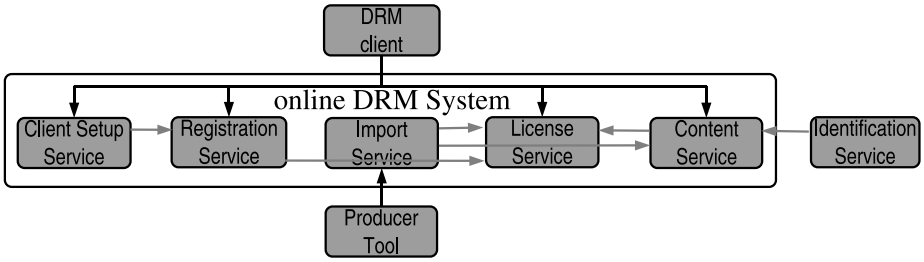

Fig. 1. Relevant DRM services

4.2 Content License and Root License

We distinguish between content licenses and root licenses as shown in figure 2.

Content licenses are enablers for a consumer to consume specific content according to some predefined set of rights. Content Licenses are not bound to and thus independent of any device. A content license can be written formally as $(\text{"}content\text{"}, Id_{license}, Id_{content}, Id_C, Id_{LS}, K^E_{content}, rights, date_{expiry})sig_{LS}$. "*content*" determines the license category, $Id_{license}$ uniquely identifies the license, $Id_{content}$ the content to which the rights apply, Id_C the license owner (the consumer) and Id_{LS} the issuing License Service; *rights* specify the associated rights. The encrypted content key $K^E_{content}$ can only be decrypted at the consumer's side with the help of a root license. A content license expires at $date_{expiry}$. The content license is signed by the issuing LS. We will often drop the word "content" as prefix when talking about content licenses.

Root licenses are enablers for a consumer to use content licenses on a specific device. Without a root license, a consumer cannot consume content on that device. Each user needs a separate root license for each device he/she wants to use. A root license is issued by a Registration Service and can be formally written as $(\text{"}root\text{"}, Id_C, Id_{RS}, Id_{lroot}, SK^E_{lroot}, date_{expiry})sig_{RS}$ which is analog to the content license. Each (registered) consumer has been assigned (but not given) one secret root key SK_{lroot}. It is encrypted in the root license of the consumer for that device such that only the TPM can get hold of it (even not the consumer himself). This key is required to decrypt the encrypted content key in content licenses.

We distinguish between public computers, semi-public computers and private devices. Public computers are used by a lot of (often occasional) users. These are for instance computers in a cybercafe or public library. Semi-public computers are used by a limited set af users. A typical example is the home computer used by all the family members. Private devices are typically used by a single person (its owner) examples are mobile phones and MP3 players. On public (and semi-public) computers, we must avoid that anyone using that computer can consume the rights of another consumer. Therefore, the consumer need to authenticate to the DRM client. A simple login/password is not sufficient, because this can be shared by multiple persons. We thus need an authentication mechanism such

that the DRM client is sure that the authenticating person is indeed the owner of the root license. Therefore, smartcards could be used: e.g. electronic identity cards, which are being issued by the governments of several countries. These electronic identity cards have the advantage that the owner is not keen to give or lend it to others and that each civilian has one. On the other hand, on private devices this smartcard authentication is not really necessary and would even be highly impractical. Smart card authentication is indicated with a dashed line in figure 2.

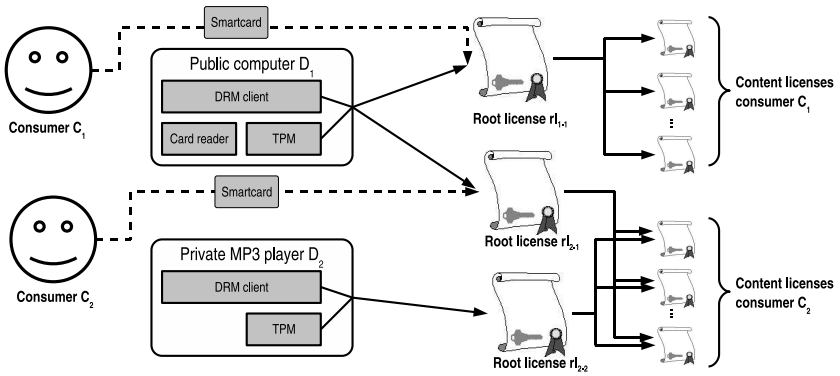

Fig. 2. A root license is required to consume DRM protected content on a specific device. A smartcard is required to consume content on (semi) public computers.

4.3 Assumptions

The different entities involved need certificates for authentication purposes, including the output devices such as display devices, which have to prove their trustworthiness. We assume that the output devices have decryption capabilities, and that the decrypted content in the output devices cannot be captured, such that the signal only travels the expected way.

The (new) methods described below must be added to the current TPM interface. The entity calling this functions must be authenticated to the TPM.

- A method $true/false \leftarrow storeKey(K_{name}, name, policy)$, which stores the key K_{name} into the non-volatile memory of the TPM. A policy *policy* is associated with it, determining what the caller of the function or other entities are able to do with the key. The caller of this function becomes the owner of the TPM stored key K_{name}. The TPM also stores the name of the key and the identity of the key owner. The latter can be an application, but still, the TPM owner must give his/her consent before this method is executed to prevent abuse.
- A method $Id_{K_{name}} \leftarrow loadKey(name)$, which loads a TPM stored key with a given name in the volatile memory of the TPM. The TPM checks the policy associated with the key before the function is executed. i.e. the caller

of this function is authorized first. A pointer to the loaded key is returned. Only the caller of this function is authorized to use this pointer.
- A method $Id_K \leftarrow decrypt(K^E, Id_{K_{dec}})$, which decrypts the encrypted key K^E with the key referred to by $Id_{K_{dec}}$. The key K_{dec} must be loaded in the TPM's volatile memory beforehand with the $loadKey()$ method. Only a pointer to the key K is returned. The key thus never leaves the TPM. The TPM authorizes the caller of the $decrypt()$ function to execute this function only after having verified the policy corresponding to K^E (see the $storeKey()$ method) and after having verified that the caller of this function is also the owner of the pointer $Id_{K_{dec}}$.
- A method $decryptAndSend(Id_K, content_i^E, Id_{output})$ decrypts the content block $content_i^E$ using the key in the TPM's volatile memory referred to by Id_K and confidentially sends it to the output device with id Id_{output} (after output device authentication). Again, Authorization is given to the caller after having checked the policy file and the ownership of the pointer.
- The method $true/false \leftarrow deleteKey(name)$ removes the key K_{name}, together with the associated data such as its policy, out of the TPM's non-volatile memory. Only the TPM owner or owner of the key (i.e. the one who executed the $storeKey(*, name, *)$ method) can get authorized by the TPM to do this.

An authentication mechanism is required such that the application can authenticate towards the TPM by proving ownership of an embedded key using white box cryptography, but without knowing the key value itself.

We think that these extensions are feasable. Firstly, TCG is planning to extent the TPM specifications with policy support (see [10] section 2.4). Secondly, if we omit the $decryptAndSend()$ method, all the proposed methods are theoretically possible in the most recent TPM specification (V1.2). In fact, besides the policy support, only the API and some access control must be added to the TPM. Thirdly, the most difficult extension seems to be the $decryptAndSend()$ method, because current TPMs do not support mass symmetric encryption. However, hardware implementation of symmetric algorithms such as AES are easy and cheap.

4.4 Protocols

In this section, the different protocols required in the DRM framework are described at a high level. The functionality of the less relevant protocols is only briefly explained. We refer to the full report (see [13]) for all details.

The properties of the communication links are indicated with the letters "I" (integrity), "C" (confidentiality) and "A" (authentication) above the communication arrows. An indirect connection is indicated with an "*". For example, $B \overset{AI}{\leftrightarrow} A$ means that both A and B need to be authenticated, and that the messages, which are sent in both directions, have to be integrity protected. Abstraction is made of the user authentication.

Content submission. The producer negotiates a contract with the Import Service IS before the producer submits the content. IS sends the content to the Content Service CS and the usage rules to the License Service LS.

DRM client installation. To install a DRM client, the Client Setup Service CSS is contacted. CSS embeds a key in the new DRM client using white-box cryptography, applies code guarding and sends the DRM client to the consumer. This key is also imported in the TPM by CSS (hidden for, but with the consent of the consumer) and will be used to authenticate the DRM client to the TPM and to DRM services. The public key of the top level CA (Certification Authority) for DRM services is also imported in the TPM by CSS using the *storeKey* method. The policy states that the key can only be changed by the CSS, but other entities can view it. The DRM client will use it to verify the validity of licenses. The security properties (e.g. code guarding type) of the DRM client can be sent to the Registration Service RS by the CSS.

Device Registration. Devices must be registered before they can consume content. If the DRM system is convinced of the thrustworthiness of the device (in fact the TPM guarding it) and the installed DRM client, the consumer will obtain a per user root license, while the key K_{TPM-C} necessary to extract the root key from the root license is imported in the TPM. The DRM client is only given permission to use this key for cryptographic operations performed inside the TPM. The per user root license key K_{lroot} is encrypted with this key. Only the TPM can thus get hold of the K_{lroot}. RS knows the id of the DRM client and the consumer because these need to be authenticated. By simply registering a device, i.e. by requesting a root license for it, C is thus able to use all his content licenses and content on that device. When a consumer registers his first device, K_{lroot} is first generated (and stored at RS by *retrieveRootKey*.

1	$C \overset{AI}{\rightarrow} RS$	$registrationRequest(cert_{TPM})$
2	$TPM \overset{A*}{\leftrightarrow} RS$	$proofSystemSecurityStatus()$
3	$TPM \overset{A}{\leftrightarrow} client \overset{A}{\leftrightarrow} RS$	$checkDRMClientSecurityStatus()$
4	RS	$K_{TPM-C} \leftarrow genKey()$ $K_{lroot} \leftarrow retrieveRootKey(Id_C)$ $K^E_{lroot} \leftarrow enc_{K_{TPM-C}}(K_{lroot})$
5	$C \leftarrow RS$	$license_{root} \leftarrow send((..., K^E_{lroot}, ...)sig_{RS})$
6	$TPM \overset{ACI*}{\leftarrow} RS$	$storeKey(K_{TPM-C}, TPM - C, \{Id_{client} : crypto\_use = yes\})$
7	RS	$store(Id_C, Id_{TPM}, date_{currrent})$
8	$LS \leftarrow RS$	$send(Id_C, K_{lroot})$

The $proofSystemSecurityStatus()$ starts a protocol already available in current TPMs. While executing $checkDRMClientSecurityStatus()$, the DRM client proves his identity by proving ownership of the embedded key (using white-box crypto). This is enough for RS to look up the security properties of the DRM client. Optionally, the TPM could be used to further enhance the protocol.

Content request. The consumer C retrieves DRM protected content $content^{WE}$. The protected content is watermarked with the transaction id to enable consumer

identification in case of abuse detection. A proof of the transaction details is agreed and stored by CS. Because the proof contains all relevant data, including the transaction id, which are signed by the consumer, this can indeed be considered as a proof. The key to decrypt the content is sent to the License Service LS for inclusion in a license.

1	$C \overset{AI}{\rightarrow} CS$	$request(Id_C, Id_{content})$
2	$C \overset{A}{\leftrightarrow} CS$	$proof \leftarrow (Id_{content}, Id_C, Id_{transaction}, timestamp)sig_{SK_C}$
3	CS	$content \leftarrow retrieveContent(Id_{content})$
		$K_{content} \leftarrow genKey()$
		$content^W \leftarrow watermark_{PK_{IdS}}(content, Id_{transaction})$
		$content^{WE} \leftarrow enc_{K_{content}}(content^W)$
4	$CS \overset{IA}{\rightarrow} C$	$send(Id_{content}, content^{WE})$
5	$CS \overset{IAC}{\rightarrow} LS$	$send(Id_C, Id_{content}, K_{content})$

License request. Once the consumer has retrieved protected content, he/she can request a corresponding license. After receiving a request, the License Service LS retrieves the rights, encrypts the corresponding content key $K_{content}$ with the per user root license key K_{lroot}. LS indeed has knowledge of both keys. This encrypted key is included in the license. K_{lroot} is generated and sent to RS if it wasn't generated beforehand.

1	$C \overset{IA}{\rightarrow} LS$	$send(Id_C, Id_{content}, Id_{licenseType})$
2	LS	$K_{content} \leftarrow retrieveContentKey(Id_{content}, Id_C)$
		$K_{lroot} \leftarrow retrieveRootKey(Id_C)$
		$K^E_{content} \leftarrow enc_{K_{lroot}}(K_{content})$
		$rights \leftarrow retrieveRights(Id_{licenseType})$
3	$C \leftarrow LS$	$license \leftarrow (Id_C, ..., K^E_{content}, rights, ...)sig_{LS}$

Content Consumption. After having obtained a root licence on the device, DRM protected content and a corresponding device license, the consumer will be able to consume the content. The DRM client verifies whether the action is allowed or not. This also includes checking the validity of the licenses. The DRM client retrieves a pointer to K_{TPM-C}, which is used to let the TPM decrypt K^E_{lroot}. K_{lroot} is loaded in the internal volatile memory of the TPM. Only a reference is returned, such that it can be used to decrypt $K^E_{content}$ in a similar way. Once the content key is known by the TPM, *client* sends the content to the TPM which decrypts it and confidentially sends it to the output device.

1	$C \rightarrow client$	$consume(content^{WE}, license, license_{root}, action, Id_{output})$
2	$client$	$actionAllowed(license, license_{root}, action, Id_{output})$
3	$client \overset{AC}{\rightarrow} TPM$	$Id_{K_{TPM-C}} \leftarrow loadKey(TPM - C)$
		$Id_{K_{lroot}} \leftarrow decrypt(Id_{K_{TPM-C}}, license_{root}.K^E_{lroot})$
		$Id_{K_{content}} \leftarrow decrypt(Id_{K_{lroot}}, license.K^E_{content})$
		$decryptAndSend(Id_{K_{content}}, content^{WE}, Id_{output})$

Identification. The identification Service IdS is the only entity that is able to identify the source of abuse when unencrypted but watermarked content is found by extracting the watermark. The embedded transactioon id can be used to retrieve the transaction details signed by the consumer.

Device Deregistration. Old devices can be unregistered to prevent further consumption of DRM content on that device. The DRM client sends a request to the Registration Service RS. RS establishes a confidential connection with the TPM and removes K_{TPM-C} (and associated data such as the policy) stored by the TPM by executing the $deleteKey(TPM - C)$ method on the TPM. Once this is done, RS locally removes the registration tuple.

Rights often are time related. The DRM client thus needs a tamper resistant clock. The DRM client is able to detect tampering with the system clock by using a tick counter and a monotonic counter, complemented with an online timing service (see the report [13] for details).

5 Analysis

Leaking secret key data. If no extra protection mechanism is present, which is the case in current computers, the content of the internal memory can be leaked by doing memory dumps, or by reading swap data. At the consumer's side, the sensitive key information is only in clear text in the shielded volatile memory of the TPM. Only the DRM client is authorized to request decryptions with the key, without ever having access to the key itself. The online DRM system also knows secret key information (e.g. K_{lroot}). Classical protection mechanisms are required here. The confidentiality of the symmetric key used by the DRM client to authenticate, depends on the robustness of the applied white-box cryptography algorithm. Keys are always transferred confidentally. It must not be possible to extract DRM keys out of the TPM.

Leaking unencrypted content. Measures must be taken by the publisher and producer to keep the content secret using classical cryptography. The only place at the consumer side where content (or the decrypted keys) resides unencrypted is in the TPM and in the output device. We assumed that the latter is sufficiently protected. Thus, even if the DRM client is broken, the consumer or an attacker cannot get hold of the content.

Spreading of recorded content. It is impossible to avoid recording of content once it leaves the output device (e.g. recording audio). This is called the analog gap, which is bridged by inserting a watermark that contains the transaction identifier. The spreading of content relies on the robustness of the watermarking scheme that is used.

Rights extension or theft. We rely on existing code guarding techniques to detect tampering with the functionality of the DRM client. If the code guarding is broken, the rights can no longer be enforced. This may allow the consumer to extend his 'rights' in an illegal way. Still, the consumer can not get hold of the content or sensitive key information. If a set of DRM clients is considered insecure, the consumers can be forced to update their DRM client when a new root license is required. Typically, the lifetime a root license will be rather short.

This is not necessarly an extra burden to the consumer, because it can be made invisible for him/her.

During the client installation, the TPM is requested to store the public key of the DRM public key of the top level CA. The certification chain verification of a root license or content license does not succeed without this public key. Thus, the consumer cannot take the encrypted keys out of valid licenses and put them into e.g. self signed licenses with more rights.

Only C can obtain a root license for his/her devices. If no consumer authentication is required before content can be consumed, C can simply give another consumer C' illegal access to content by placing a root license on C''s device. Therefore, it is important that the number of root licenses that C can have at the same time is limited and that C has to authenticate to the DRM client on (semi) public computers before he/she can consume content.

The clock tampering detection mechanism avoids that consumers consume content after expiry of the content license or root license. When a root license is found expired, the Registration Service RS will request the TPM to remove the associated symmetric key. The registration record stored by the Registration Service will also be deleted after expiry.

If a DRM client is found compromised or insecure, RS will try to establish a connection with the TPM in order to remove K_{TPM-C} the next time the device connects to the internet. A new root license will only be issued after having updated the DRM client. However, this cannot always be enforced and consent of the TPM owner will be required.

With the current technologies, the services can be convinced of the trustworthiness of the system and the DRM client. The DRM keys can be stored by the TPM. However, cryptographic functions are performed by software. Keys and output of these cryptographic functions are stored in the internal memory, which can be obtained by the consumer. The DRM extensions presented in this paper thus offer a considerable increase towards secure DRM. In current solutions, the security is often based on code obfuscation. Knowing the hidden algorithms enables consumers to extract the content if a valid license is present. This framework does not rely on secrecy of algorithms, but only on secrecy of keys.

If a trustworthy DRM client is replaced by a non trustworthy one, the latter will not succeed in getting authorization by the TPM when trying to consume content, because it lacks the white-box crypto embodiment of the key needed to authenticate..

6 Comparison with Existing Technologies

More and more, DRM is appearing in new products such as Compact Disks and MP3 players. All companies owning the technologies try to hide as much details about the inner working as possible. Recently, we have seen Sony's DRM technology on CDs being critisized for creating hidden files on the consumer's system and for running secretly processes that can compromise the system's

stability. These files are hard to remove without losing access to your CD-drive. If we look at Microsoft's Windows Media DRM and Apple's Fairplay (used in iTunes and iPod), we see the same: they do not offer much information about the inner working and trust on the secret keeping of algorithms that are used by their DRM clients. E.g. Windows Media DRM uses code obfuscation to hide the algorithm that derives the DRM client key. Fairplay has similar problems. Sooner or later, these algorithms will be discovered, resulting in the DRM technologies being broken.

One important aspect in this paper is that we provide an open DRM system; anyone may know the inner working. We can compare the evolution in DRM with the evolution of cryptography in general where we saw a change from hiding algorithms to hiding keys. Our framework uses several building blocks that can be replaced if necessary. At the moment, only Windows Media DRM offers limited updates for its DRM clients, however if the algorithms are revealed, more than a simple update is required. Another important aspect of this framework is the binding of rights to users instead of to devices.

The presented framework needs more hardware support, which is indispensable to have secure DRM. The key information must not be exposed, which is only possible with hardware or operating system support. Microsoft is working on NGSCB (see [9]), which should offer operating system support similar as the operating system based solution presented in [11]. In the case of DRM, the consumer must be seen as a potential adversary. Therefore, we think that it is hard to combine open source and DRM support in operating systems. The presented solutions indeed requires extra hardware support, but can still be used on open source operating systems such as Linux.

7 Conclusion and Future Work

In this paper, we presented a flexible DRM framework that is more secure than existing technologies. To have a secure DRM technology that cannot be broken by consumers or other attackers, we need protection against memory space snooping. This can be done by hardware or operating system support. This paper presented a hardware based solutions, whereof we think that these allow a greater degree of openess and simplicity than the operating system approach. The hardware based solution needs some extensions to the current TPM (v1.2). We argued that the current TPM offers a good basis such that minimal extensions satisfy to allow secure DRM systems. This paper can be seen as a proposal towards TCG to extend the TPM for DRM purposes. Of course, appropriate watermarking, code guarding and white box cryptography is required. These technologies can only be seen as blocks in a complete DRM system. Only the most essential key information is hidden while at the same time the functionality of the DRM client is protected. This paper tried to identify the crucial protocols. Extra services and protocols can be added. The paper is high level, but hopefully, it will help in developping mature DRM.

As part of future work, stateful licenses will be taken into account. These limit the number of times certain actions on the content can be performed by

the owner of the license. Conditional anonymity will be added, such that the identity of both producer and consumer is not revealed in the case of normal usage. This will be applied in the domain of e-health. Delegation of rights and DRM client revocation will also be tackled. This should allow the owner of rights to lend, give or sell part of these rights to others.

References

1. M. J. Atallah, E. D. Bryant, M. R. Stytz. A survey of Anti-Tamper Technologies. 2004.
2. Mauro Barni and Ingemar J. Cox and Ton Kalker and Hyoung Joong Kim. Digital Watermarking, 4th International Workshop, IWDW 2005, Siena, Italy, September 15-17, 2005, Proceedings.
3. S. Chow, P. Eisen, H. Johnson and P.C. van Oorschot. A White-Box DES Implementation for DRM applications. In Proceedings of ACM CCS-9 Workshop DRM 2002, volume 2595 of Lecture Notes in Computer Science, pages 1-15. Springer-Verlag, 2003.
4. S. Chow, P. Eisen, H. Johnson and P.C. van Oorschot. White-Box Cryptography and an AES implementation. SAC 2002 - 9th Annual Workshop on Selected Areas in Cryptography, Aug.15-16 2002, St. John's, Canada. Proceedings (revised papers): pp.250-270, Springer LNCS 2595, 2003.
5. Bill Horne, Lesley Matheson, Casey Sheehan, and Robert E. Tarjan. Dynamic Self Checking Techniques for Improved Tamper Resistance. ACM Workshop on Security and Privacy in Digital Rights Management, pages: 141 - 159, 2001.
6. William Ku and Chi-Hung Chi. Survey on the Technological aspects of Digital Rights Management. ISC 2004: 391-403, 2004.
7. Windows Media DRM 10 cracked?. http://www.engadget.com/2005/02/01/windows-media-drm-10-cracked/, 2005.
8. S. Michiels, K. Verslype, W. Joosen, B. De Decker. Towards a software architecture for DRM. DRM '05: Proceedings of the 5th ACM workshop on Digital rights management, 2005.
9. Microsoft Next Generation Secure Computing Base http://www.microsoft.com/resources/ngscb/default.mspx.
10. G.J. Proudler. Concepts of Trusted Computing. IEE Professional Applications of Computing Series 6, 2005.
11. J. F. Reid, W. J. Caelli. DRM, Trusted Computing and Operating System Architecture. 2005.
12. Trusted Computing Group. TCG TPM Specification Version 1.2 Revision 94, https://www.trustedcomputinggroup.org/specs/TPM, 2006.
13. K. Verslype, B. De Decker. A Flexible and Open DRM construction. KULeuven dept. computer science, technical report, 2006.