Hash-based Counter Scheme for Digital Rights Management

Mikko Löytynoja, Tapio Seppänen
MediaTeam Oulu
Department of Electrical and Information Engineering
University of Oulu, Finland
{mikko.loytynoja, tapio.seppanen}@oulu.fi

Abstract

This paper describes a counter scheme that uses hash functions to count how many times the user is allowed to play protected content in a DRM-enabled player. The proposed basic scheme can be used in scenarios where the user cannot be assumed to have online connection. We discuss the weaknesses of the proposed scheme and present alternative to the basic scheme, which increases the security of the counter.

1. Introduction

Content such as music and video is increasingly in digital form. This increases the risk of illegal copying, as it is very easy to make perfect copies from digital content, whereas when copying analog content the quality of each generation is worse than the previous one. Digital rights management (DRM) tries to find a solution to this problem inside a triangle set by technology, economics and law.

DRM is not just copy protection of content; it enables new business possibilities as the users can be granted many kinds of usage rights to content, e.g. limited number of usages, limited period of usage, and different cost for viewing & printing. These rights can be used to create flexible billing schemes, e.g. user pays certain amount if he/she prints document less than 10 times, but after that the price of printing is cheaper. Limiting period of usage enable network based video rental business, where user is allowed to play movie for one day. These rights are usually defined using rights expression languages (REL), such as ODRL [1], XrML [2] and MPEG-21 REL [3].

The biggest vulnerability of any DRM system is that any software-based system, if not running on tamperproof hardware, can be modified and thus the protection mechanism can be circumvented. Encryption protects from these kinds of attacks to a certain level and watermarking can be used to protect the content after decryption. But if the attacker has the key available, e.g. by purchasing one license to the content, he/she can then freely distribute the content to other users using cracked players. Frequent updating and patching the security holes in players can be used as countermeasures against these attacks. A trusted computing platform could be used to prevent users from modifying the player; the trusted platform module can calculate hash of the player software and verify that it has not been tampered. It still remains to be seen when and what kind of trusted computing platforms will emerge. [4] Additionally, it is always possible to record the content in analog form while it is presented to the user. Only way to protect against this DA/AD attack is to insert something to the content itself such as watermark.

The rights expression languages define what the user can do with the content and under what conditions, but they do not define how these rights are enforced. One of these conditions that are presented in RELs and in DRM solutions from e.g. Microsoft, RealNetworks, Adobe, is a limited number of usages. Since the most present or upcoming DRM products have the feature to restrict number of times content can be consumed, it is very important to create secure way to count the usages. In the XrML specification examples the condition of a limited number of usages is implemented with a reference to some external service and the ODRL does not specify how the limit is implemented in practice either. The implementation of this condition is much more complex than what it first seems. In a scenario, in which the user is online during the content consumption, a specific server can be used to count the remaining usages. But this is not a feasible solution in a scenario where the user cannot be assumed to be online all the time, e.g. mobile use or with a dial-up connection to the network.

In DRM systems the user's terminal must be considered as hostile, since the user benefits, if he/she can break the protection and e.g. get unlimited playback of the content. How to count the remaining usages on a user's terminal, if the user is able to access and modify all the data stored on that terminal? Encryption is not feasible solution, because the keys must be stored on that same device, hence they are available to the user. There are ways to obfuscate encryption keys into application, so that it is very hard to retrieve them as shown in [5][6][7]. It is still possible to attack against the obfuscation [8] and the computational complexity increases when using obfuscation [7], which might restrict its usage in mobile devices.

In this paper we propose a hash-based counter scheme that can be used to count the number of remaining usages. The basic counter scheme that is presented in Section 2 can be used in an offline scenario. In Section 3 we discuss the vulnerabilities in the proposed scheme and present improvement to basic scheme when using NTFS file system. Section 4 presents some experimental results of the implementation in mobile phone. Section 5 concludes the discussion.

2. Basic Counter Scheme

The license for content can limit the number of times that the user is allowed to consume the content. Our proposed basic hash-based counter scheme can be used to count the number of usages left in a scenario, in which we assume that the user does not necessarily have an online connection and that the licenses which describe what the user can do with the content are protected by digital signatures and the signatures can be verified with public-key certificates. We also assume that the player software is trusted, i.e. it is not hacked and it will adhere to the license terms.

In our current implementation of the DRM system, we use XML-based licenses. The test system is presented in more detail in Section 3. The proposed scheme is based on a scheme originally developed as micropayment mechanism proposed simultaneously by [9][10][11]. These micropayment schemes enabled the user to have a number of tokens that could be used to pay very small amounts.

The scheme uses hash function, which takes variable length input and turns it to fixed length output. The hash function is a one-way function, i.e. it is very hard to guess its input if only the output is given. Examples of cryptographically strong hash functions are MD5 [12] or SHA-1 [13]. When the license is first created, a random string t_0 is generated. The hash function h is the applied n times recursively

$$t_{i+1} = h(t_i)$$

for $i = 0, 1, \ldots, n-1$, where n is the number of times that the user is allowed to consume the content.

The license is divided into two parts, one of which is digitally signed by the DRM license server and the other one is not. The t_0 and n are stored in the unsigned part of the license and the t_n and n are stored in the signed part of the license. When the user consumes the content, the player applies the hash function to t_0 and replaces the value stored in the unsigned part of the license with t_1 and updates the counter value in unsigned part to n-i etc. The remaining amount of usages left in the license can be validated by hashing the value in the unsigned part n-i times and checking that the output equals t_n.

```
<License>
<Signature>loDet09DTwestFS</Signature>
<SignedPart>
    <Count count="5">2Q9ksZ4jd8</Count>
    …
    </SignedPart>
<UnsignedPart>
    <Counter count="3">uCkZm6vuk=</Counter>
</UnsignedPart>
</License>
```

Figure 1. XML license showing counter.

Figure 1 shows an example of the XML license file used to mark the number of usages in our test system. The Signature element contains signature information about the SignedPart of the license. The value of Count element is the t_n and it also has an attribute, which defines what the value of n is. The value of n is needed to check that the count attribute, which is n-i, in Counter element is not greater than initial number of usages. This ensures that the user cannot modify the t_i in unsigned part and calculate how many times the hash function needs to be performed to get t_n and so increase the value of counter.

The difference of the proposed scheme compared to the original micropayment schemes is that counting is reversed in the DRM counter scheme. In the micropayment scheme the client sends the last hash value to the vendor, signed by a trusted third party. After that the client sends the previous hash values to the vendor when he/she wants to pay something, as shown below.

$$t_n, t_{n-1}, t_{n-2}, \ldots$$

The DRM counter functions by calculating the next hash value and replacing the previous one with the new one. Since the previous counter values must be permanently erased, the DRM counter is more vulnerable against attacks.

3. Improving the Basic Counter Scheme

The proposed scheme needs quite strict restrictions in order for it to be secure. The biggest weakness of the scheme is that previous counter values must be erased to prevent the user from resetting the counter to an earlier value.

The proposed scheme prevents the user from increasing the number of usages left in the license. In a case where the counter is simply a value stored in the license, a malicious user could freely change the value to whatever he/she wishes. In our scheme, the user would need to find the value that produces the hash functions output in order to increase the counter, which is very hard if the hash function used is cryptographically strong. Although the basic scheme is not very secure, it is more secure than storing the counter value in plaintext, since the value cannot be increased. Additionally, the scheme is not computationally heavy, since the calculation of remaining usages requires only a few hash operations.

In the following, we present what restrictions the scheme sets to the scenario, in order to prevent the user from attacking the counter.

3.1. Replay Attack

The basic counter scheme prevents the user from increasing the counter value, but the user is still able to keep the counter from decreasing. The basic scheme assumes that the user is offline and hence the counter must reside on the user's terminal. The easiest way to attack the proposed counter scheme is by a method that we call replay attack.

Since the user is usually able to access the data stored on the terminal, he/she can make a backup copy of the licenses. After the backup, the user plays the content normally and the DRM counter decreases the remaining usages normally. When the license has run out of usages, the user restores the license from the backup. This undoes all the changes that the DRM enabled player has made to the licenses and the license can be used again to play the content. In other words the counter never decreases.

The replay attack can be used in all scenarios where the counter must reside on the user's terminal and it is impossible to protect against it, if the user has write access to all the data stored in the terminal. One solution to this attack is to store the counter in the server, but this would prevent offline use of protected content. Trusted computing platform could also be used to check that the user has not modified the license files or to provide storage space where only the player software is able to store data. [14] The latter solution requires that the hardware and the operating system support this kind of features.

3.2. NTFS File Last Access Attribute

When using NTFS file system each file has three date attributes; creation, modification and last access. [15] The last access date is changed each time the file is moved, copied, backed up or modified etc. This fragile nature of the attribute could be used to make replay attack more difficult. The player software stores, in addition to counter data, the date and time when the counter was last decreased. This same date and time are stored in file's last access date attribute. Next time the user tries to play protected content, the player checks that the two dates match, if they differ the license has been tampered.

This method can naturally be attacked. Easiest way is to use file management application that enables to change the last access date of the file. If the date stored inside the license is obfuscated the user cannot simply check what is the correct date and set it as the last access date attribute. So the user must first get the utility software and he/she must also store the current date of each license file before playing them. This at least makes the replay attack more tedious.

The drawback of this method is that the last access attribute could be changed by some application even if the user has not been tampering the license, so the user should not even read the contents of the license files.

This same kind of approach could be used with file systems if there is date attribute that is changed each time the file is copied. Preferably that date attribute would not change if the file is read, as sometime is the case with NTFS's last access date.

4. Experimental results

We have implemented the proposed scheme in our DRM architecture [16]. The computational cost of the hash function is very low even with a player running in mobile phone. We implemented the basic counter scheme to a player running in Nokia 660, which was released Q4 2003. On average, with the hash functions implemented with Java 2 ME in Nokia 6600, it is possible to process 200 kb of data with MD5 and 100 kb with SHA-1 using block sizes from 8 bytes to 8192 bytes. So, calculating the counter value takes less than millisecond.

Unfortunately, the implementations of the secure counters in current DRM products are not publicly available information, which makes it hard to compare their implementations to ours.

5. Conclusions

Without a secure platform a perfect copy control may be impossible. Our pragmatic goal is therefore to create technology that complicates content copying in varying degrees. In this paper, we presented a counter scheme, which uses a hash function to prevent the user from increasing the number of remaining usages.

The scheme is feasible in a scenario, where users cannot be assumed to have an online network connection, but it is wished that content consumption be limited to a certain number of usages. In this offline case the counter data must be stored on user's terminal, which must be considered as hostile environment. The proposed scheme enables counting usages and prevent user from increasing the counter value. The drawback of the scheme is that counting the usages on the user's terminal is open to replay attacks, if the user is able to access all the data on the terminal. This can be countered with limiting the user's access to license files with the help of the operating system and secure hardware. Moving the counter to a server limits the use scenario to online use, which is not feasible in many cases. We also presented a method that utilizes NTFS's last access date attributes. Because the attribute is changed each time the file is copied, it can be used to make replay attack harder.

6. Acknowledgments

The financial support from the National Technology Agency of Finland (Tekes), Graduate School in Electronics, Telecommunications and Automation (GETA), and the Stardust project consortium is gratefully acknowledged.

7. References

[1] The Open Digital Rights Language, http://www.odrl.net/

[2] The eXtensible Rights Markup Language, http://www.xrml.org/

[3] DeMartini T., Wang X. & Wragg B. (ed.) *MPEG-21 Rights Expression Language.* MPEG-21 Working Documents Parts 5 & 6, March 2003.

[4] Becker E., Buhse W., Günnewig D. & Rump N. "Trusted Platforms, DRM and Beyond". In *Digital Rights Management: Technological, Economic, Legal and Political Aspects.* Springer LNCS, v. 2770, pp. 178-205.

[5] Chow S., Eisen P., Johnson H. & van Oorschot P.C., "White-Box Cryptography and an AES Implementation", In *Proceedings of the Ninth Workshop on Selected Areas in Cryptography,* Springer-Verlag, 2003, pp. 250-270.

[6] Chow S., Eisen P., Johnson H. & van Oorschot P.C., "A White-Box DES Implementation for DRM Applications", In *Proceedings of 2nd work ACM Workshop on Digital Rights Managements (DRM 2002)*, Springer-Verlag, 2003 November 18, pp. 1-15.

[7] Link H.E. & Neumann W.D., "Clarifying Obfuscation: Improving the Security of White-Box Encoding". In *Cryptology ePrint Archive, Report 2004/025.* http://eprint.iacr.org/2004/025.pdf

[8] Jacob M., Boneh D. & Felten E., "Attacking an Obfuscated Cipher by Injecting Faults", In *Proceedings of 2nd work ACM Workshop on Digital Rights Management (DRM 2002)*, Springer-Verlag, 2003 November, pp. 16-31.

[9] Anderson R.J., Manifavas C. & Sutherland C. NetCard – "A Practical Electronic Cash Scheme". In *Security Protocols (1996)*, Springer LNCS, v. 1189, pp. 49-57.

[10] Pedersen T.P. "Electronic Payments of Small Amounts". In *Security Protocols (1996)*, Springer LNCS, v. 1189, pp. 59-68.

[11] Rivest R.L. & Shamir A. "PayWord and MicroMint: Two Simple Micropayment Schemes". In *Security Protocols (1996)*, Springer LNCS, v. 1189, pp. 69-87.

[12] Rivest R.L. "The MD5 message-digest algorithm". *Internet Request for Comment (RFC 1321)*, April 1992.

[13] National Institute of Standards and Technology (NIST) "Secure Hash Standard". In *FIPS Publication 180*, May 11 1993.

[14] Cram E. "Next-Generation Secure Computing Base: Development Considerations for Nexus Computing Agents" Microsoft, October 2003,

[15] Microsoft, "File Times", *Platform SDK: Windows System Information,* MSDN Library, http://msdn.microsoft.com/library/default.asp?url=/library/en-us/sysinfo/base/file_times.asp

[16] Löytynoja M, Seppänen T & Cvejic N "Experimental DRM architecture using watermarking and PKI". In *proceedings of 1st International Mobile IPR Workshop: Rights Management of Information Products on the Mobile Internet*, HIIT Publications, Helsinki, Finland, August 27-28 2003, pp. 47–52