

# Building secure software-based DRM systems

Gisle Grimen, Christian Mönch, Roger Midtstraum

Department of Computer and Information Science  
Norwegian University of Science and Technology  
{grimen, moench, roger}@idi.ntnu.no

## Abstract

Software-based DRM systems promise fast access to a large market. But for a long time it has been assumed that software-based DRM systems are inherently insecure and should not be used for high quality content. We present a way to build secure software-based DRM systems. After analysing the weaknesses of current DRM systems we devise a strategy for the construction of software-based DRM systems that are capable of withstanding a large variety of attacks.

## 1 Introduction

Digital distribution of high quality entertainment over networks such as the Internet has an tremendous potential. The benefits to both consumers and content owners are huge when compared to existing distribution systems. Immediate access to content with low-cost delivery is one of the new benefits Internet-based distribution brings. The ubiquitous availability of powerful processing and communication devices make it possible for everybody to distribute digital media, allowing dishonest people to distribute content without compensating the content owners. While these new technologies have the potential to open up new markets, the risk of abuse makes copyright owners reluctant to use them.

One way to tackle this problem is to add technical measures to the distribution and consumption process that ensure that digital media are only used in accordance to the rights the user has, e. g. view unlimited, copy only three times, don't redistribute. Systems that implement the handling and supervision of these rights are usually summed under the term *Digital Rights Management* (DRM) systems. While DRM systems do in principle manage all rights a user has to a certain media document, it is not certain that DRM systems can actually enforce these rights. The DRM system can only behave as expected if the platform it is executed on behaves as expected. If the platform or the execution environment of a DRM system or the system software itself is changed, the system might fail to enforce the restrictions on the use of the media document.

A common approach to protect a DRM system against tampering is to use a hardware-based protection, often implemented in set-top-boxes. But this solution has a number

---

*This paper was presented at the NIK-2006 conference; see <http://www.nik.no/>.*

of disadvantages. The biggest disadvantage of hardware based protection systems is inflexibility. A network-based distribution technology requiring special hardware can not leverage the tremendous amount of consumers already connected to the Internet. Special hardware has to be developed, manufactured and sold before a hardware-based DRM system can be employed. This requires a large investment from the service provider and increases time to market. These are major disadvantages in a fast moving market. Additionally, hardware-based systems are expensive for customers. At a time where a lot of pirated content is available on the Internet, hardware-based solutions have a hard time creating value for the consumer.

An alternative to hardware-based DRM systems are purely software-based DRM systems that can be executed on general purpose computers. Their advantage over hardware-based systems is that they can cheaply be distributed to the customers over digital networks and that they, except for the initial design, do not create additional per-installation costs. Most users would prefer a legal way to easily access content without huge initial costs or a long term commitment. A software-based DRM solution would immediately unlock a huge market and could attract a large number of potential consumers that up until now obtained their content from the Darknet [1]. The problem with software-based DRM systems is that they are assumed to be insecure. While software-based DRM systems have been broken, this is also true for hardware-based systems [13, 2].

In this paper we present a strategy for the construction of software-based DRM systems that are far more secure than existing software-based systems. The remainder of this paper is structured as follows: in the next section we analyse the main problem of DRM systems. Then we analyse how software-based DRM systems can be attacked and give some examples of successful attacks. Based on our analysis we present a strategy for creating a secure software-based DRM system and describe a sample implementation of such a system.

## **2 The principal weakness of DRM systems**

DRM systems have to perform two conflicting tasks. They have to prevent the user from accessing media documents, while at the same time they have to allow access to the same media documents. More specifically, DRM systems have to enable certain operations on the media, for example presentation, while preventing other operations like copying. Performing an operation on a media document implies that whatever technical measure has been put into place to prevent the user from accessing the media, has to be removed—or, circumvented—by the DRM system in order to present the media to the user.

Figure 1 illustrates the operation of the DRM system in a typical client setting. The media document is protected by encryption while it is transported to the player software, which runs on the user's host. In the player, the DRM system removes the protection by using an appropriate decryption key. The media document is now available in plaintext to the player software. Depending on the licence, the player might now perform an operation on it, for example presentation, which yields data that is accessible to the user, for example video frames.

The removal of the protection has to be performed by the player software on the user's host. That means the player software is executed in an environment that is completely controlled by the user. Consequently, the user can change the software in such a way that it removes the protection but does not enforce the usage licence, giving the user unrestricted

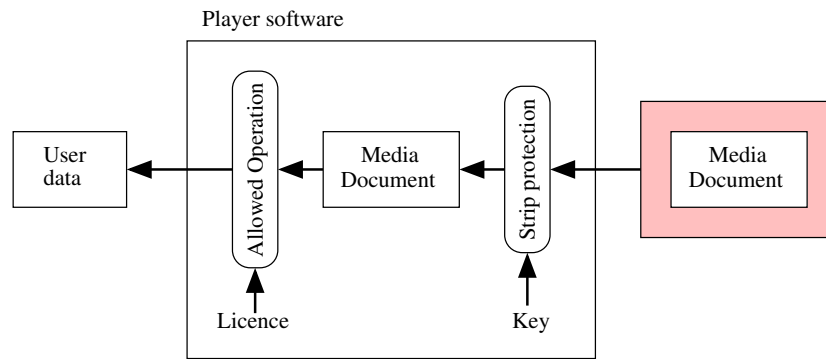


Figure 1: Handling protected media in the player software

access to the media document. The problem with software-based DRM systems is that it is difficult to ensure the integrity of the player software in a foreign environment. This is known as the malicious host problem [11]. The malicious host problem states: if software is executed on an uncontrolled computer, correct execution can not be ensured.

## 2.1 Encryption is not the silver bullet for DRM systems

Strong encryption is often used in DRM systems to transport media documents over insecure communication channels. It is a common misconception to believe that this encryption protects the media document during the whole presentation process and that accessing an encrypted media document is as difficult as breaking the encryption. In a DRM system the encryption does not have to be circumvented by an attacker. If a media document is encrypted, the DRM system has to remove the encryption before processing the media document. This means that the decryption keys have to be known to the player software, i. e. be stored in the memory of the player software. Furthermore, the media document will be stored, at least partially, in plaintext in the memory of the player software. Because the player program is executed in an environment that is controlled by the user, the user can extract all data that is processed by the player program, including the decryption keys and the media document in plaintext.

An effective software-based DRM system therefore requires additional mechanisms to ensure that the sensitive information being processed by the system is protected against extraction. While these mechanisms should preferably be as secure as the encryption itself, existing systems use mechanisms that are vastly inferior to the level of protection that encryption can deliver.

## 3 Attacks on software-based DRM systems

In order to prevent attacks on software-based DRM systems, one has to be aware of the threats that such a system faces. We have therefore analysed and classified the most common attacks on software-based DRM systems.

The primary assumption of all attacks is that the player software is executed on a malicious host. This means that the user who executes the player software has the possibility to analyse it and to use the knowledge gained to attack it. An attack might be the extraction of sensitive data or the modification of the player software so that it

does not enforce the rights anymore. Every sensible attack requires two things: collecting information about the player software, and an intellectual understanding of the software.

In a malicious host environment there is no way to prevent the collection of information about the software. Information can be collected by static or dynamic analysis of the program. Static analysis is based on disassembling and possibly decompiling the software of the DRM system. Although several methods have been developed to prevent automated disassembly [6, 17] none of them can withstand a skilled attacker. The development of anti-disassembling techniques have lead to the development of more robust disassembling techniques [16]. Dynamic information is obtained by executing the player software in a debugging environment. Different techniques have been developed to sabotage debuggers [6, 23], but this has only led to the development of more sophisticated and robust debuggers, for example [19].

After the information gathering, the next step in an attack is to gain understanding of the player software. Different methods, usually referred to as obfuscation techniques, have been developed to make the intellectual analysis of disassembled or decompiled code more difficult [7, 25, 5, 20]. But again, none of these techniques can hide the program's algorithms from a skilled attacker. Nevertheless, they might increase the time needed to obtain the required knowledge about the player software, which is necessary to perform a successful attack.

### 3.1 Classes of attacks on software

We distinguish four classes of attacks that are possible on player software which are illustrated in figure 2. These attacks are classified according to whether they modify the program and whether they use information gathered during runtime of the program.

|                  |         |                         |                             |
|------------------|---------|-------------------------|-----------------------------|
| Used information | Dynamic | Spying Attack           | Dynamic Modification Attack |
|                  | Static  | Analysis Attack         | Static Modification Attack  |
|                  |         | No                      | Yes                         |
|                  |         | Modification of program |                             |

Figure 2: Classes of attacks

The four different attacks, analysis attack, spying attack, and the static and dynamic modification attacks are explained in more detail below. They can be used isolated or in combination in order to circumvent the DRM system in the player software.

#### 3.1.1 Analysis attack

In an analysis attack the attacker analyses the player software in order to extract sensitive information like decryption keys or key-generating algorithms. Apple's iTunes

application is vulnerable to an analysis attack, which can uncover the secret algorithm used to generate the repository key [21].

### *3.1.2 Static and dynamic modification attacks*

If the attacker has gained enough knowledge about the player software, he can perform modifications of the program in order to disable or circumvent the protection mechanism contained in it. The player software could, for example, be modified to ignore licences and grant the user full access to the unprotected media document. A successful modification attack was performed by adding a module to an mp3-player software that created an unprotected copy of every song, including restricted songs, that were played by the player software [15].

We distinguish between static modification attacks and dynamic modification attacks. Static modification attacks are performed entirely before the player software is executed. A software-patch is a typical example for a static modification attack. Once the player software is executed no further modification is performed. Dynamic modification attacks modify the player software during its execution. They use information about the current state of the executing player software. It should be noted, that dynamic modification attacks require sophisticated virtualisation environments that are probably not available to an average user.

A typical means to prevent modification attacks is the calculation of checksums over the instructions of the player software in memory. However, the problem with checksum algorithms is that they are executed in the same environment as the player software and are therefore also susceptible to modifications. In order to increase robustness against modification, one could use several overlapping checksums as described in [3, 12]. A different approach which calculates a checksum over the result of the executed instructions is presented in [4]. This checksum algorithm is more difficult to attack, because its expected result is not immediately obvious.

Checksum algorithms are especially weak against dynamic attacks. Wurster et al. present a dynamic attack on checksum algorithms in [26, 24] that is based on the observation that checksum algorithms perform an atypical operation, i. e. they perform data access on program code.

### *3.1.3 Spying attack*

Spying attacks are based on monitoring the memory image of an executing player software in order to extract sensitive information from it. A successful spying attack might, for example, extract decryption keys or media data in plaintext from the memory image.

At present, spying attacks have received little attention, mostly because there are easier ways to extract sensitive information from player software. Microsoft's DRM solution attempts to prevent simple spying attacks by storing sensitive data in a non-sequential way in memory [10], making it difficult to locate data by a pattern based search, e. g. by searching for MPEG headers. Madou et al. discuss the problem of spying attack and show its potential by attacking a newly developed software watermarking technique [18]. In a malicious host setting it is not possible to prevent or even detect spying attacks.

## 3.2 Examples of attacks on DRM systems

### 3.2.1 Fairplay

Fairplay is Apple's DRM solution used to protect Apple's online music business and is integrated into the iTunes application running on the user's computer. The Fairplay protection mechanism is in itself fairly simple and relies heavily on encryption [14]. The protection solution consist of three encryption keys, the master key, the user key and the repository key, shown in figure 3. The AAC audio in the MP4 file is encrypted with the master key. The master key is stored in the same container file, encrypted with a randomly chosen user key. The user key is stored along with the user information on apples server. In addition the user key is provided to the iTunes application and stored in its key repository along with the identification information of the file.

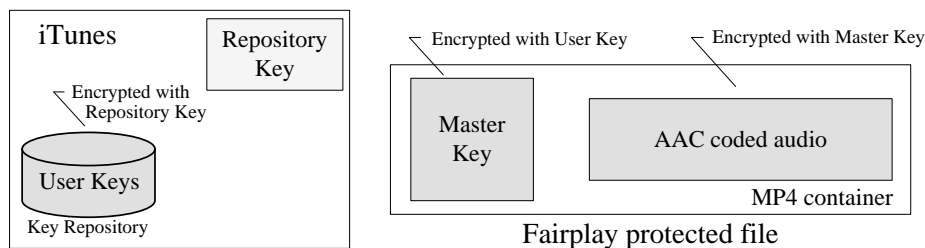


Figure 3: Fairplay encryption hierarchy

The iTunes key repository is encrypted with the repository key. The repository key is at the top of the encryption hierarchy and can as such not be protected by encryption. Instead, the repository key is stored in the iTunes application in form of a secret algorithm. The iTunes application is able to obtain the repository key as needed by executing the secret algorithm. The security of Fairplay is therefore completely dependent on the ability to keep the secret algorithm hidden from attackers. As the repository key is hard-coded into the application in form of the secret generation algorithm, an analysis attack aimed at finding out how the repository key is generated is in itself enough to completely circumvent the protection mechanism.

The details about the techniques employed by Apple to prevent the secret algorithm from becoming public are not exactly known. Reverse engineering of the iTunes client for Windows has revealed that the repository key is derived from four factors: the serial number of the first hard drive, the system BIOS version, the CPU name and the Windows Product ID [21]. This information is used as an input to a hash function to generate the repository key. Attempts by Apple to fix the Fairplay protection mechanism after successful breaches have been to alter the way iTunes generates the repository key.

### 3.2.2 Microsoft DRM

The protection scheme used in Microsoft DRM is similar to the one used in Fairplay. A simplified overview of the DRM-system is shown in figure 4. The media file is encrypted with a content key. The content key is again encrypted by a client specific set of keys and stored in the licence. The licence is stored in a licence store at the user's computer [10].

The licence can be accessed and the content key obtained by the *blackbox* (see figure 4), which contains the client specific keys. The interesting part of Microsoft's DRM

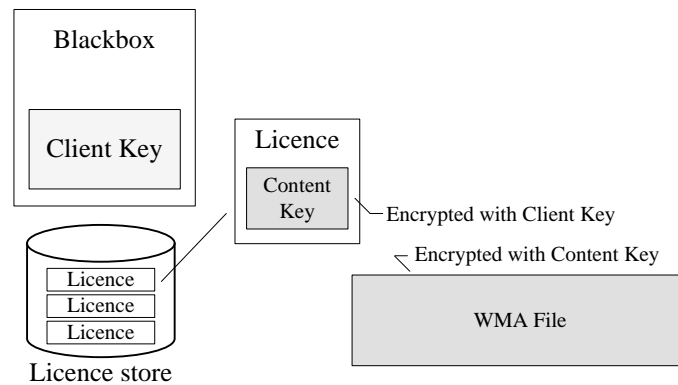


Figure 4: Microsoft DRM

compared to Fairplay is that the blackbox is individualised. Individualisation means the blackbox is created specifically for that computer and will not work on any other computer. In theory, individualisation has also the effect that identification of the client specific keys can not easily be automated, ensuring that an intellectual analysis of each blackbox is needed. At present the individualisation strategy of Microsoft DRM is not known.

A person under the pseudonym Beale Screamer published an attack against version two of Microsoft DRM [22]. The attack took advantage of a weakness in the design of the blackbox, making the blackbox itself extract the client specific keys. With an efficient way to obtain the main secrets from each program, only some general reverse engineering was needed in order to find the correct licence and remove the encryption from the protected file.

The attack used by Beale Screamer is a classic modification attack. It uses the actual code written for the protection mechanism in a new way together with some additional code in order to remove the encryption from the protected file. By partly using the protection mechanism's own code it effectively bypasses the individualised protection designed to prevent a global breach of the mechanism.

### 3.3 Classes of breaches

If attacks have successfully revealed sensitive information from two different DRM systems, the severity of the resulting breaches might not be identical. Depending on the structure of the DRM systems and the selected protection mechanisms, the revealed sensitive data might unlock a large number of media documents or just a small fraction of a single media document. According to the amount of media documents lost as a result of a single intellectual attack, we distinguish four levels of breaches: global breach, repository level breach, document level breach, and subdocument level breach.

**Global breach** All media documents that are protected by the attacked system are accessible. Examples for this breach are the CSS-hack, the attack on Apple's Fairplay, and the Microsoft DRM attack.

**Repository level breach** All media documents that are contained in a repository, i. e. all media, which belong to a single customer, are accessible. With a flawless blackbox design breaches of Microsoft's DRM would be restricted to this level.

**Document level breach** A single media document is accessible.

**Subdocument level breach** A fraction of a single media document is accessible.

It is obvious that a system that allows only breaches on the subdocument level is more resilient because it would require a large number of individual intellectual attacks to gain unlimited access to even one media document. Since the extraction of data from player software can not be prevented, a software-based DRM system should be structured to allow only breaches of the subdocument class.

## 4 Designing a secure software-based DRM system

A software-based DRM system that is capable of preventing unauthorised access to media data, has to withstand the three attacks described in section 3.1. Consequently such a system has to fulfil the following three requirements:

**No secrets in the player** The player program should not contain secret information, because all information contained in the player will eventually be revealed by an analysis attack. Secret information might include decryption keys and algorithms.

**Verify integrity** In order to prevent modification attacks the system has to be able to verify that the player is unmodified and behaves as expected. In addition it should ensure that changes in the underlying operation system do not alter the player's behaviour.

**Prevent automated extraction of data** In order to prevent spying attacks the system should prevent automated extraction of sensitive information from the player's memory. In other words, extraction of sensitive information should require repeated intellectual attacks.

Based on these three requirements we identified the following design principles for software-based DRM systems: downloadable protection algorithms, controlled data release, and active protection of the player.

### 4.1 Downloadable protection algorithms

A software-based DRM system has to remove all secrets from the player to withstand analysis attacks. Nevertheless, the player will need access to secret information and algorithms when it processes media data. If code can be downloaded into the player during its runtime, then the inactive player does not need to contain any secrets.

When protection mechanisms are downloaded into the player, they become vulnerable to the same attacks as the player. This can be prevented by repeatedly downloading new protection mechanisms, because any sensible modification of code requires an intellectual understanding of the code. Gaining this understanding takes time. If the protection algorithms are replaced frequently enough, one can be sure that the protection algorithms are unmodified. A precondition for this time-based protection is that the downloaded code pieces are sufficiently distinct from each other to require a new intellectual attack.

## **4.2 Controlled, piecewise data release**

The system should release as little information to the player at a time as possible. This has two important consequences. First it allows the system to enforce the download of code to the player in exchange for additional data delivery. Second, if the released portions of the data are independently secured, then the partial release limits the impact of a breach. This means that one breach would only reveal a single, individually secured part of the media document.

## **4.3 Active protection of the player**

The downloadable code has to contain protection algorithms, which perform three basic tasks. The first task is to ensure that the protection mechanisms are executed in the player. To ensure their execution, they have to be designed in such a way that their results are crucial for access to the media data. The second task is to ensure that the player system has not been modified. The third task is to restructure the player in such a way that any attack that might have worked on the previous player will not work on the restructured player. This provides timely separated player instances that require individual intellectual attacks in order to extract information from them. The restructuring thus protects a player against automated extraction of data. Preferably, the restructuring is performed every time a new part of data is released to the player.

# **5 A secure software-based DRM system**

We have designed and built a secure software-based DRM system for streaming media [8]. The system consists of three components: a player, a streaming server and a security server. The player decrypts and decodes the media stream in the user's environment. The streaming server provides the player with a protected media stream and may or may not be in a trusted environment. The security server is located in a trusted environment and forces the player to cooperate in order to gain access to the media stream.

To force the player to cooperate with the security server, we encrypt the media stream with a unique key every few seconds, referred to as a media key. By releasing keys one by one every few seconds, the security server ensures that the player is forced to cooperate during the complete presentation process. The continued cooperation is important for two reasons. First it enables the unique creation of previously unknown protection mechanism, whose download to the player is enforced at runtime. Second, it enables that at no single point in time the player contains information that unlocks the complete media stream.

Although no secret is stored in the program, we still need to be able to securely obtain the media keys from the remote server. To individually and independently secure the transport of media keys from the security server to the player, we use an algorithm that generates unpredictable data in the player. Contrary to Apple's Fairplay algorithm that uses unique, but static input, we use a dynamic algorithm with unpredictable input to generate a random number at runtime, referred to as a transport key. We securely exchange the transport key with the remote server through public key encryption and use it as a short time secret in order to securely obtain the next media key from the remote server.

The downloadable protection algorithms, referred to as Mobile Guards, are created at the security server and downloaded at runtime to the player as a prerequisite for obtaining

a media key. To verify the integrity of the player, the protection mechanisms contain a randomly created checksum algorithm entangled in the Mobile Guard [9]. During execution of the Mobile Guard, the checksum algorithm is executed and computes a unique value, based on parts of the memory image of the player. The result of the checksum is sent back to the server, which checks the result. The use of a random checksum algorithm ensures that the Mobile Guard itself is executed and that an unmodified player is present in memory.

To shield sensitive information the Mobile Guard performs runtime obfuscation tasks on the player's memory. The goal of runtime obfuscation is to invalidate an attacker's knowledge about the player's structure and mode of operation, making it infeasible to automatically extract sensitive information from the player. Runtime obfuscation consists of three steps. The first step is to shuffle sensitive information around in memory, while applying a simple mask to the data, changing its appearance. Secondly, basic blocks of instructions are diversified and shuffled around in memory. Diversification consists of changing instructions of the basic blocks while keeping the semantics of the basic blocks equivalent. After sensitive information has been altered and code and data have been shuffled around, instructions are repaired to reflect the new location and structure of the data.

Runtime obfuscation and randomised checksum algorithms together ensure that an unmodified player is executing and receiving the media keys. As a result our system is protected from the dynamic attack against checksums that was presented in [26, 24].

## 6 Conclusion

We analysed the principal weaknesses of DRM systems and devised a way to construct a secure software-based DRM system. The resulting strategy allows construction of DRM systems that will withstand the common attacks. The resulting systems are protected against analysis attacks, ensure the integrity of the player software, and prevent automated data extraction. Although the solution still relies on secrets in the memory of the player software, these secrets allow only subdocument level access and are completely independent of each other. To breach the system a new intellectual attack has to be performed on each individual segment. The effort involved in obtaining even a single media document is prohibitively expensive in terms of time and intellectual resources, because automating the individual attacks is not possible. Compared to existing systems, which are susceptible to global breaches and repository level breaches, a system which is designed according to our strategy will provide superior security.

## References

- [1] P. Biddle, P. England, M. Peinado, and B. Willman. The darknet and the future of content distribution. In *Proceedings of the 2002 ACM Workshop on Digital Rights Management*, November 2002.
- [2] L. Jean Camp. Drm: doesn't really mean digital copyright management. In *CCS '02: Proceedings of the 9th ACM conference on Computer and communications security*, pages 78–87, New York, NY, USA, 2002. ACM Press.

- [3] Hoi Chang and Mikhail J. Atallah. Protecting Software Code by Guards. In *DRM '01: Revised Papers from the ACM CCS-8 Workshop on Security and Privacy in Digital Rights Management*, pages 160–175. Springer-Verlag, 2002.
- [4] Yuqun Chen, Ramarathnam Venkatesan, Matthew Cary, Ruoming Pang, Saurabh Sinha, and Mariusz H. Jakubowski. Oblivious Hashing: A Stealthy Software Integrity Verification Primitive. In *IH '02: Revised Papers from the 5th International Workshop on Information Hiding*, pages 400–414. Springer-Verlag, 2003.
- [5] Stanley Chow, Yuan Gu, Harold Johnson, and Vladimir A. Zakharov. An Approach to the Obfuscation of Control-Flow of Sequential Computer Programs. In *ISC '01: Proceedings of the 4th International Conference on Information Security*, pages 144–155. Springer-Verlag, 2001.
- [6] Frederick B. Cohen. Operating system protection through program evolution. *Computers and Security*, 12(6):565–584, October 1993.
- [7] Christian Collberg and Clark Thomborson. Watermarking, tamper-proofing, and obfuscation: tools for software protection. *IEEE Trans. Softw. Eng.*, 28(8):735–746, 2002.
- [8] Gisle Grimen, Christian Mönch, and Roger Midtstraum. Software-based copy protection for temporal media during dissemination and playback. In *the 8th International Conference on Information Security and Cryptology*, volume 3935 of *LNCS*. Springer, December 2005.
- [9] Gisle Grimen, Christian Mönch, and Roger Midtstraum. Tamper Protection of Online Clients through Random Checksum Algorithms. In *the 6th International Conference on Information Systems Technology and its Applications*, LNI, May 2006.
- [10] Tobias Hauser and Christian Wenz. DRM Under Attack: Weaknesses in Existing Systems. In *Digital Rights Management - Technological, Economic, Legal and Political Aspects*, pages 206–223. Springer-Verlag, 2003.
- [11] Fritz Hohl. Time Limited Blackbox Security: Protecting Mobile Agents From Malicious Hosts. In *Mobile Agents and Security*, pages 92–113. Springer-Verlag, 1998.
- [12] Bill Horne, Lesley R. Matheson, Casey Sheehan, and Robert Endre Tarjan. Dynamic Self-Checking Techniques for Improved Tamper Resistance. In *Digital Rights Management Workshop*, pages 141–159. Springer-Verlag, 2001.
- [13] Andrew Huang. Keeping Secrets in Hardware: The Microsoft Xbox Case Study. In *Revised Papers from the 4th International Workshop on Cryptographic Hardware and Embedded Systems*, pages 213–227. Springer-Verlag, 2003.
- [14] Jon Lech Johansen. Fairplay. In *So sue me - blog*, May 2005. <http://nanocrew.net/2005/05/13/fairplayno/>.
- [15] John P. Mello Jr. Aol pulls plug on napster pirate plug-in. In *TechNewsWorld*, February 2005. <http://www.technewsworld.com/story/40699.html>.

- [16] C. Kruegel, W. Robertson, F. Valeur, and G. Vigna. Static disassembly of obfuscated binaries. In *In Proceedings of USENIX Security 2004*, pages 255–270, August 2004.
- [17] Cullen Linn and Saumya Debray. Obfuscation of executable code to improve resistance to static disassembly. In *CCS '03: Proceedings of the 10th ACM conference on Computer and communications security*, pages 290–299, New York, NY, USA, 2003. ACM Press.
- [18] Matias Madou, Bertrand Anckaert, Bjorn De Sutter, and Koen De Bosschere. Hybrid static-dynamic attacks against software protection mechanisms. In *DRM '05: Proceedings of the 5th ACM workshop on Digital rights management*, pages 75–82, New York, NY, USA, 2005. ACM Press.
- [19] J. Maebe, M. Ronsse, and K. De Bosschere. DIOTA: Dynamic instrumentation, optimization and transformation of applications. In *Proceedings of the 4th Workshop on Binary Translation*, 2002.
- [20] Toshio Ogiso, Yusuke Sakabe, Masakazu Soshi, and Atsuko Miyaji. Software tamper resistance based on the difficulty of interprocedural analysis. In *The Third International Workshop on Information Security Applications (WISA 2002)*, pages 437–452, August 2002.
- [21] Andrew Orłowski. itunes drm cracked wide open for gnu/linux. seriously. In *The Register*, January 2004. [http://www.theregister.co.uk/2004/01/05/itunes\\_drm\\_cracked\\_wide\\_open/](http://www.theregister.co.uk/2004/01/05/itunes_drm_cracked_wide_open/).
- [22] Beale Screamer. Microsoft's digital rights management scheme - technical details. sci.crypt, 2001. <http://osiris.978.org/brianr/crypto-research/ms-drm/ms-drm.htm>.
- [23] Andres Torrubia and Francisco J. Mora. Information Security in Multiprocessor Systems Based on the X86 Architecture. *Computers & Security*, 19(6):559–563, 2000.
- [24] Paul C. van Oorschot, Anil Somayaji, and Glenn Wurster. Hardware-assisted circumvention of self-hashing software tamper resistance. *IEEE Trans. Dependable Sec. Comput.*, 2(2):82–92, 2005.
- [25] Chenxi Wang, Jonathan Hill, John Knight, and Jack Davidson. Software Tamper Resistance: Obstructing Static Analysis of Programs. Technical Report CS-2000-12, Dept. Computer science, University of Virginia, Charlottesville, VA, USA, 2000.
- [26] Glenn Wurster, Paul C. van Oorschot, and Anil Somayaji. A generic attack on checksumming-based software tamper resistance. In *IEEE Symposium on Security and Privacy*, pages 127–138, 2005.