Singapore Management University
Institutional Knowledge at Singapore Management University

Research Collection School Of Information Systems School of Information Systems

8-2007

Enhanced Security by OS-Oriented Encapsulation in TPM-Enabled DRM

Yongdong WU
Institute for Infocomm Research

Feng BAO
Singapore Management University, fbao@smu.edu.sg

Robert H. DENG
Singapore Management University, robertdeng@smu.edu.sg

Marc MOUFFRON
EADS Secure Networks

Frederic ROUSSEAU
EADS Secure Networks

DOI: https://doi.org/10.1007/978-3-540-79499-8_37

Follow this and additional works at: https://ink.library.smu.edu.sg/sis_research

Part of the Information Security Commons

Citation

WU, Yongdong; BAO, Feng; DENG, Robert H.; MOUFFRON, Marc; and ROUSSEAU, Frederic. Enhanced Security by OS-Oriented Encapsulation in TPM-Enabled DRM. (2007). *Information Security and Cryptology: Third SKLOIS Conference, Inscrypt 2007, Xining, China, August 31 - September 5*. 4990, 472-481. Research Collection School Of Information Systems.
Available at: https://ink.library.smu.edu.sg/sis_research/397

This Conference Proceeding Article is brought to you for free and open access by the School of Information Systems at Institutional Knowledge at Singapore Management University. It has been accepted for inclusion in Research Collection School Of Information Systems by an authorized administrator of Institutional Knowledge at Singapore Management University. For more information, please email libIR@smu.edu.sg.

Enhanced Security by OS-Oriented Encapsulation in TPM-enabled DRM

Yongdong Wu[1], Feng Bao[1], Robert H. Deng[2]
Marc Mouffron[3], Frederic Rousseau[3]

[1] Institute for Infocomm Research, Singapore
{wydong,baofeng}@i2r.a-star.edu.sg
[2] Singapore Management University, Singapore
robertdeng@smu.edu.sg
[3] EADS Secure Networks, France
{marc.mouffron, frederic.rousseau}@eads.com

Abstract. The Trusted Computing Group (TCG) defines the specifications for the Trusted Platform Module (TPM) and corresponding trust mechanisms that allow a TPM-enabled platform to run only authenticated software. For example, the operating system (OS) can use the facilities provided by the TPM to authenticate a Digital Rights Management (DRM) application before allowing it to run. However TCG does not provide any clear specification on what kind of software can be regarded as trusted and hence be authenticated. In fact it is unlikely to draw a clear line between the software that should be authenticated and those should not. For instance, it may be controversial to authorize debugger for developing binary codes and/or Internet browser for running applets on TPM platform. This leaves a grey area where even authenticated software may be exploited for malicious usage.
This paper investigates the security of DRM applications in a reasonably relaxed scenario where users have larger purview. It presents two attacks: abuse attack and injection attack where some reasonably authenticated software can be exploited for stealing protected contents. In the abuse attack, an attacker uses an authenticated debugger to monitor the internal state of a DRM application for the purpose of violating the access privilege in the application. In the injection attack, an adversary is able to make malicious modifications on an original DRM application at will. These two attacks demonstrate that it is not straightforward to impose DRM in a TPM-enabled system. To counter the attacks, we provide the OS-encapsulation scheme which encrypts the DRM application such that only the genuine OS can start the DRM application. Our scheme can be viewed as an enhancement of security for TPM-enabled DRM in a loose but more practical environment, where people are still allowed to use the debugger, web browser, etc.

1 Introduction

General-purpose computer platforms provide the convenience for developing and running software applications. However, it also allows a malicious user to tamper with and possibly control any software component at will. Software tampering not only affects the software industry in general, it can also impede content delivery services and compromise enterprise systems. Hence, many software vendors employ software protection technologies to counter such threats. Unfortunately, it has been generally accepted in the information security research community that software alone cannot provide an adequate foundation for building a high-assurance trusted application [1]. Hardware-assisted protection [2] is a promising solution for strong software security.

In the past decade, cryptographic hardware modules have played such trusted-bootstrap role on behalf of secure wireless hand-holds for governmental users or military software defined radios or on behalf of bank cash dispensers or on behalf of bank card payment terminals. For instance, ARM (`http://www.arm.com`) released `Trustzone`® as its solution for trusted platforms, and many mobile platforms nowadays (mobile phones, bank card payment terminals) are hardware platforms that run an OS over ARM-embedded micro-controllers.

To bring the security feature of handhold device to general purpose platforms such as personal computers, the Trusted Computing Group (TCG) [3] defines the specification for a security chip called the Trusted Platform Module (TPM) and its related software interfaces. In the TCG specification, the TPM is designed to provide end-user machines with a minimal but essential hardware base for end-user side security. For instance, there are TPM-enabled schemes in enhancing the security of smartcard [4], attestation [5–7], and privacy protection [8–10]. Since it is impossible to modify the TPM-enabled OS without being detected, the TPM can be employed to defeat sophisticated attacks such as the substitution attack [11, 12] which invalidates software integrity protections (*e.g.*, [13, 14]). In addition, Felten [15] highlights the possibility that a DRM application can utilize the TPM to protect data such that only cooperating applications are able to consume the protected content.

Although the TPM can potentially be used in many applications, its protection mechanism is still not satisfactory [15–18]. For instance, Anderson [16] criticized TPM-based DRM applications as treating users in an unfriendly way. Additionally, in a DRM application, a traitor[4] may misuse a genuine software to compromise the built-in access control in the application.

1.1 Our contribution

TPM platform restricts the access right of virus, applications and users, hence it provides some valuable security functions. However, we need to consider the loose scenarios where users are given reasonably more authority. In practice, it is hardly acceptable to employ a very restrictive confine on software usage, *e.g.*, disabling loaders including software debugger and/or network browser.

This paper investigates the security of the TPM-based DRM where some reasonably authenticated software such as debugger can be used, and proposes

[4] A traitor is a user who modifies his authorizations illegally. For example, the traitor may change his read permission into write permission

the abuse attack and the injection attack. In the abuse attack, a malicious user makes use of a trusted application such as a debugger to mediate the interactions between the victim program and the OS. On the other hand, given that a user is allowed to sign and certify his own program on the TPM-enabled platform, the injection attack enables him to run any program locally at his discretion. Both attacks allow the malicious user to violate the content protection objective of DRM. To counter these attacks, it is necessary to verify the owner/signer of the application, and to establish the application's authenticity. To achieve this, we propose to encrypt the DRM application and start the decrypted application by the authenticated operating system only, thus prevent both abuse attacks and injection attacks. By the two attacks and their countermeasures, we demonstrate that TPM does not guarantee DRM security automatically and easily. Many issues have to be solved before we can claim that the trusted computing based DRM has better security.

This paper is organized as follows. Section 2 presents an overview of the TPM. Section 3 introduces the model of TPM-enabled DRM. Section 4 describes a basic TPM-enabled DRM scheme, followed by two attacks on the scheme. Section 5 proposes our techniques to the TPM-enabled DRM scheme to counter the attacks. Section 6 concludes the paper.

2 Overview of TPM

TCG is an industry working group which aims to establish industry standards for trust and security in computing platforms. It was founded in 1999 by HP, IBM, Intel and Microsoft, and there are now close to 150 member companies. The TCG's TPM is one example of hardware chip that aims at providing the root of trust of a platform. It is especially applicable to platforms that would otherwise not have any hardware security module. In the TCG specification for TPM, a TPM chip provides on-chip key pair generation using a hardware random number generator, as well as RSA cryptographic functions. The TPM communicates with the system CPU on a non-system bus, and acts only under the control of the system CPU and the policies codified in the OS and TPM-enabled software applications.

2.1 TPM component

To provide the trusted computing platform (TCP) with root of trust, protected storage, key generation and attestation, a TPM module includes: (1) processor, (2) hardware Random Number Generator (RNG), (3) cryptographic engine, (4) non-volatile memory, (5) real-time clock, (6) tamper control circuitry, and (7) non-volatile storage space.

The TPM processor controls the functions and sequencing of the entire TPM module. It moves data between the system processor and the internal TPM memory, sequences the cryptographic engine, and helps offload the RSA computation from the general purpose system processor.

The RNG is the source of seed numbers for the cryptographic engine's encryption and decryption functions, and for the generation of RSA key pairs. It also provides random values for generating signatures.

Secure storage stores sensitive data such as the permanent Endorsement Key (EK) and Storage Root Key (SRK). EK is bound to a unique platform by the manufacturer, while SRK is tied to the TPM owner who takes ownership of the TPM. All other cryptographic keys are protected by the SRK. An Attestation Identity Key (AIK), generated by TPM, is a special key used for generating digital signatures. Each TPM can have as many AIKs as it needs for the sake of anonymity.

The real-time clock provides tamper-proof date stamping for authentication and attestation processes. By monitoring the voltage, clock frequency and other environment parameters with tamper control circuits, the TPM can effectively prevent illegal access to sensitive information, such as by tampering the TPM with a low voltage supply.

2.2 Transitive trust

In the trusted platform, the initialization and management functions in the TPM allow the owner to turn its functionality on and off, reset the chip, and take ownership of the chip. This group of functions provides strong separation of what can be done at BIOS boot time and what can be done at normal run-time, so that sensitive operations such as reading the endorsement key cannot be performed by malicious applications trying to compromise the TPM. Concretely, the trusted boot functions compute hashes of configuration information throughout the boot sequence, and store the hashes in Platform Configuration Registers (PCRs). If a TPM-enabled computer is reset, all the PCRs are reset too. But at any other time, the PCRs can only be extended. When a TPM-enabled system is started, a secure booting sequence will be performed as shown in Fig.1. The execution units consists of the Core Root of Trust for Measurement (CRTM), the Master Boot Record (MBR), boot sector, boot code, boot manager, the OS loader, and the OS. Each unit in the process will have its integrity measured before execution control is handed over to it. If any of the integrity measurements is invalid, all the subsequent units will not run. Only if all the units are intact, the application will be verified and executed.

3 Model of TPM-enabled DRM

In the paper, we focus on the security of TPM-enabled DRM applications on the general computer other than the dedicated DRM device since the latter is lack of extensibility, flexibility and inter-operability.

3.1 DRM Architecture

The underlying motivation for having DRM applications is that digital objects should remain under the control of their creators, rather than the owner of the

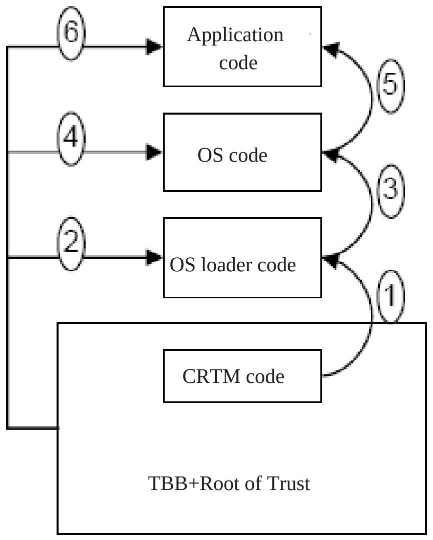

Fig. 1. Transitive trust applied to system boot from a static root of trust[19]. Trusted Building Blocks (TBB) are the parts of the Roots of Trust that do not have shielded locations or protected capabilities. Normally these include just the instructions for the Root of Trust for Measurement (RTM) and TPM initialization functions (e.g. reset).

platform containing the objects. One major threat to DRM is the tampering of the DRM application and its platform since a traitor can modify them so as to circumvent the DRM protection. Currently, a lot of commercial DRM systems (e.g., such as `Microsoft Windows RMS`, and Adobe `Web Buy`) and standards (e.g., [20–22]) are available. Fig. 2 illustrates their architectures which are as follows.

– An author generates digital content such as video file.
– A DRM-enabled application will create a rights-protected version of the file and assign usage rights at the content server. A portal server provides a Web site so that the consumer can preview, purchase and download content that interests him.
– A license server will be responsible for the enrollment and certification of trusted entities, and for licensing right-protected content.
– Finally, the consumer's DRM client software interacts with the DRM server, and renders the content according to the consumer's assigned privileges. The consumer may distribute the content to others without violating the digital right.

Although the general DRM architecture and workflow are applicable, the security depends on the assumption of a trusted environment. In lieu of a fully trusted environment, a TPM-enabled DRM application can be used to provide trust to the content owner. The model for such an application is described in the next subsection.

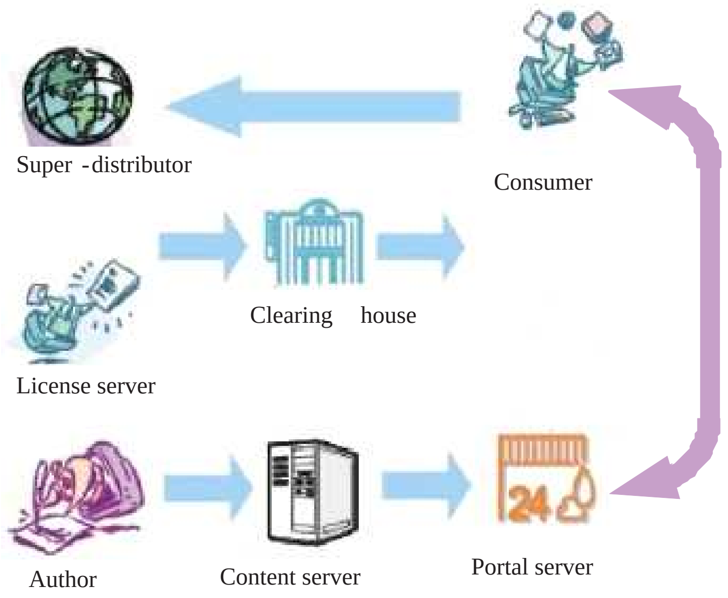

Fig. 2. General DRM architecture.

3.2 Security model

Although TCG specifications describe the technology on how to authenticate a software, it has never been explicitly described, or even discussed, what kind of software should be allowed to be authenticated. For the practical application of TCP, there is a big gap on how and who will be responsible for the software verification and validation. Indeed, it is not going to be an easy issue. From the application viewpoint, it is unfriendly to exclude all the common used software tools (e.g., debugger and Internet browser), and may result in potential market lose. To balance between right-restriction and user-friendliness, we consider a not very restrictive situation of TCP, and argue that the assumption of such a situation is very reasonable and practical.

In the system diagram Fig. 3, TPM and OS constitute the trusted platform which is compliant with the TCG specification, and supports a trust transitive process, as mentioned in Section 2. A genuine software application **P** is signed by its producer (or a trusted third party) who has a certificate issued by a CA. The CA's public key is trusted by the OS. To authenticate **P**, the OS will verify its signature before running it. Note that the OS does not use TPM integrity measurements on **P**, but the OS itself will have its integrity verified by way of TPM integrity measurements as mentioned in Section 2.2. The DRM Software **P** must be loaded by a trusted OS or a trusted debugger. According to the DRM requirements, we assume that any user has the following capabilities,

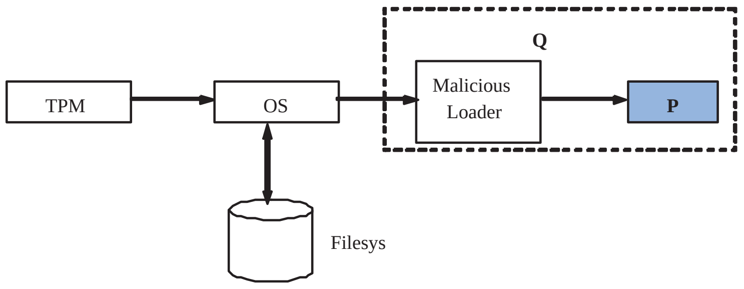

Fig. 3. System diagram. **P** is the target DRM software, and **Q** wraps **P**. **Q** will be loaded by a malicious loader such as a compromised debugger.

A_1: `able` to load the targeted program **P** in the trusted platform with his own loader. For example, an authenticated Internet browser is able to run a applet **P** which is a DRM application for rendering protected data stream. Similarly, a debugger such as `Microsoft Visual Studio` can be used in TPM environment.

A_2: `able` to intercept/modify the messages between **P** and the OS by using a malicious loader. For example, a TPM application developer should be able to debug his own software related to TPM functions. In other words, a malicious user can always "sit" between OS and DRM application so as to monitor **P**.

A_3: `able` to insert his own root certificate into the OS. By doing this, any user is able to develop and sign any software including modified application and run it on the platform.

A_4: `unable` to have the power/resource to reverse-engineer the target software **P** to gain knowledge of critical algorithms or secrets such as a cryptographic key. Otherwise, the attacker can write his own software with the reverse-engineered code and then render the protected content.

In a general computer platform, a user owns the platform and all software running on the platform including **P**. It is acceptable that a trusted general computer has some but not all limitations on the users. We argue that it is reasonable to assume user's capabilities A_1–A_3 for the following reasons:

– Although it is controversial about the function of TPM, a TPM-enabled computer must have an open structure in terms of software and hardware,

otherwise, why does a user buy expensive computer other than cheap TV setbox?
- A TPM-enabled OS prevents a user from running an "unapproved" application, but it is unacceptable to deprive the user the right of developing his own software. Hence, the platform shall allow the user to load applications that are approved by themselves.
- As argued in [17] in terms of protecting fair use rights, users should become the root of their own certification trees and authorize various devices to view purchased content.
- It is not uncommon in commodity platforms that a user is able to install his own root certificates. For example, in browser applications such as the `Microsoft Internet Explorer`, a user can import his own root certificates into the existing set of trusted root certificates.

A_4 is mandatory although it is believed to be impossible in theory. Fortunately, the software designer can increase the hacker's cost with the technologies (*e.g.*, diversity, anti-tracing, anti-disassembling, self-modifying code, and code-encryption) such that the hacker gives up the hacking goal. We can also enhance the security of software if the application code is encrypted all the time except the execution period on TPM platform.

In summary, the security model disables to start virus-infected softwares, but enables user to develop his own software, runs debugger and browser for any application but DRM application. Meanwhile, the DRM application can be only started by the designated loader such as Microsoft `explorer`. Hence, this model can enforce DRM policy without reducing user's reasonable right.

4 Basic TPM-enabled DRM: Scheme & Attack

Following the model in Section 3.1, this section introduces a basic TPM-enabled DRM idea derived from [15]. Then it presents two attacks on the basic scheme such that a traitor can obtain unauthorized access.

4.1 Basic TPM-enabled DRM Scheme

Although a TPM-enabled platform provides a trusted environment with PCR reports to detect "unapproved" software, its application to DRM is not straightforward if security and inter-operability are important. For example, Felten [15] described the TPM-enabled software inter-operation principle which can be applied to design a basic TPM-enabled DRM as follows.

- The digital file is encrypted with an authorized DRM program **P** such that the digital content is under the full control of **P**.
- If another program **Q** wants to consume the file, the only way is to request **P** to decrypt it in a TPM-enabled environment and transfer the decrypted content, but **P** maybe reject the request.

– The protected media will be decrypted on a TPM-enabled platform and used in a controllable way.

Therefore, **P** must be tamper-proof and authentic before executing if A_2 and A_4 are true. As a result, the traitor can not violate the DRM access by tampering with the software. However, basic TPM-based DRM is vulnerable to the following abuse attack and the injection attack.

4.2 Abuse Attack

With the assumption of trustworthy hardware and software, the TPM-enabled platform ensures that each application is valid when it runs. That is, no untrusted software can execute on the platform. However, the traitor can combine two or more approved programs in a certain way to realize his malicious intent. In particular, if an authorized software can load and execute some other program, as in the case of a typical debugging tool (*e.g.*, `Microsoft Visual Studio`, `SoftICE`^{TM} and `IDApro`^{TM}), the traitor can use it to access protected content as follows.

– Start the debugger. Since the debugger is genuine and is signed by the producer, it can be started successfully.
– Load application **P** with the debugger. A_1 allows for this.
– Execute **P** within the debugging environment. Since the debugger has higher execution priority than **P**, the traitor is able to read all the data in the address space of **P**, including the decrypted content.

Therefore, the traitor has easily compromised the TPM-enabled DRM protection by executing the targeted application within a debugger. Presently, Internet browser such as `Internet Explorer` or `Firefox` is able to run web-based applet, it can be misused to launch abuse attack too.

4.3 Injection Attack

According to assumption A_3, any user can insert his certificate as a root certificate into the OS. This means that any software signed by the traitor can pass the verification process described in Subsection 2.2. In this case, the traitor can initiate the injection attack to compromise the content protection mechanism of the platform. Concretely, the traitor will

(1) Install his certificate as a root certificate in the OS. A_3 allows for this.
(2) Modify program **P** by injecting functions into **P** or injecting entries in **P**'s function import tables, resulting in a malicious wrapper **Q**. For example, the traitor can replace or modify the action module of menu-item "`save`", such that it dumps all the decrypted content to memory.
(3) Sign **Q** with the private key corresponding to the newly installed certificate.
(4) Run **Q** as an authorized program. Since **P** is wrapped by **Q**, **P** is started implicitly. Now, since **P** is able to decrypt the protected content, the wrapper **Q** will be able to read the decrypted content.

(5) Invoke the injected function to dump the protected content to disk. The traitor can do this by activating the tainted menu-item "`save`" for example.

In the above injection attack, the traitor wraps an authorized software **P** and produces a new program **Q**. This attack does not require the traitor to know the internal structure of **P**. He only needs to tamper with the code partially to have the decrypted content dumped to disk.

In addition, if a user can not install his certificate, but can initiate abuse attack, he can integrate the above two attacks so as to violate the DRM efficiently, in other words, he can create the modified software **Q** and then run **Q** in the debugger or browser environment.

4.4 A naïve countermeasure

Since the basic scheme is vulnerable to the attacks addressed in Subsections 4.2 and 4.3, the security strength of DRM application should be enhanced.

A straightforward enhancement is based on PKI. Specifically, all the TPMs have different private keys (*e.g.*, EK) but the same public key. Any DRM software should be encrypted with the public key such that the protected code can be decrypted by the trusted platform only. Its weakness is that it will incur inconvenience in case of TPM upgrading.

5 OS-oriented Encapsulation Scheme

The present OS-oriented encapsulation scheme defines a new format of applications. It uses the architecture mentioned in Section 3 since the architecture is employed in many DRM commercial systems, but encapsulates the DRM application with the new format so as to match A_1−A_4 in Section 3 for sufficient flexibility and security.

5.1 Scheme description

In the proposed scheme, we intend to meeting three objectives: (1) DRM application can be loaded by the designated OS loader only; (2) allow the honest user to develop his software; (3) able to defeat injection attack and abuse attack from malicious users. To this end, the scheme includes 4 modules: configuration, encapsulation, installation and DRM enforcement. Concretely,

Configuration A Certificate Authorities (CAs) issues digital certificates to OS vendors and DRM vendors. Each OS vendor, for example Microsoft, has a pair of (public key e, private key d) for each of its OS products. The OS vendor signs the OS with its authenticated public key.

Encapsulation After a provider develops its DRM application, it will encrypt the application with the public key of the public key of destination OS. In order to guarantee the installation, each module (*e.g.*, dynamic link library) will be encrypted independently. Suppose that $\mathbf{P}$ includes l modules $m_i, i = 1, 2, \ldots$ and will be run in an operating system whose public key is e. The provider will select the software key k, and produce

$$\mathbf{C} = enc_e\mathbf{P} = \{Enc1_e(k), Enc2_k(m_1), \cdots, Enc2_k(m_l)\}. \tag{1}$$

where $Enc1(\cdot)$ is an asymmetric encryption function, while $Enc2(\cdot)$ is a symmetric encryption function respectively. Afterwards, the providers signs the encrypted own DRM application $\mathbf{C}$, and sends the ciphertext and the signature to the user. If $\mathbf{P}$ can run a multiple of operating systems, the DRM vendor encrypts the software key k in Eq.(1) with each public key of the OS.

Installation The installation process targets for setting up a trusted OS and DRM application in a TPM-enabled environment. Fig.4 illustrates the steps of installation. Specifically,

- when the OS is being installed, the installation software (such as BIOS hard-code) will initiate remote attestation with the vendor server $\mathbf{V}$. This means that the installation software will use the TPM remote attestation facilities to remotely authenticate to $\mathbf{V}$. (This is similar to OS activation over the Internet, which is currently required by Microsoft Windows XP).
- $\mathbf{V}$ securely sends the private key d of the OS to the TPM. This private key d shall be protected by the TPM and stored in sealed storage.
- After obtaining the OS package locally or remotely, the TPM-enabled platform must verify its authenticity with the digital certificate of the OS vendor. Only if the OS package is authentic, OS and other OS-related modules can be installed as usual.
- To install a DRM application $\mathbf{P}$, TPM system will obtain the ciphertext $\mathbf{C} = Enc_e(\mathbf{P})$, make sure that $\mathbf{C}$ is produced by the application provider, and install $\mathbf{C}$ directly without decryption.

DRM enforcement When the DRM application is being executed, a designated loader such as `explorer.exe` in Windows OS package will send a key request to the TPM. TPM verifies the integrity of the OS and the loader, and then decrypts the secret key d to `explorer.exe`. With d, the loader decrypts the encrypted application $\mathbf{C}$ into memory so as to obtain the memory mapping of $\mathbf{P}$. Then, $\mathbf{P}$ will be executed normally, e.g., read the content and enforce DRM protection.

5.2 Security Analysis

When all the modules of $\mathbf{P}$ are intact, the DRM application can be loaded into process memory for the DRM enforcement. Since the DRM code is encrypted in

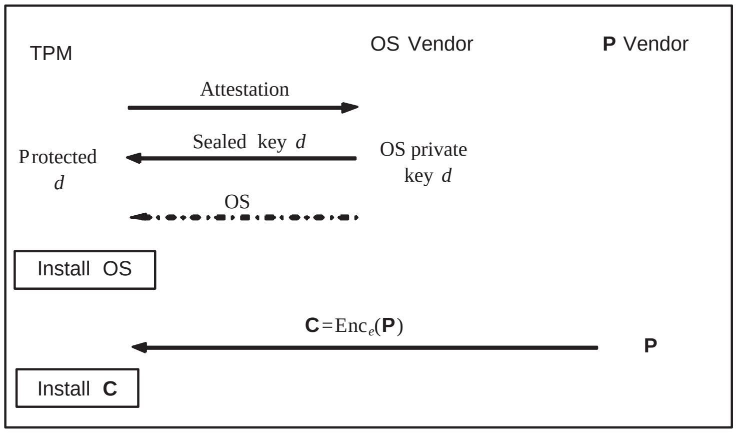

Fig. 4. Installation process. TPM is equipped with some BIOS-like hard codes for attestation when it is manufactured. The dash line for OS downloading means that OS can be obtained via other channels such as CD-ROM.

hard disk, a hacker can not statically reverse-engineer **P** so as to start injection attack directly. However, the hacker may attempt to break the DRM protection in the following ways.

– *Recording the decrypted content via the output device.* For example, voice recording or screen dumping. Fortunately, the quality of the pirated content is of low quality and result in less threat to the content industry, thus this kind of violation is out of our scope.
– *Extracting the secret key of OS.* In the installation stage, the OS provider will testify the authenticity of the TPM based on the manufacture information. As long as the TPM hardware is secure, the OS private key will be kept secret.
  Additionally, to reduce the risk associated with d being disclosed in memory, we can further split d as $d = d_1 \oplus d_2$ with secret sharing. The share d_1 is embedded into the OS directly with some software tamper-resistance technology [23–27], and the other share d_2 is downloaded from the vendor during the *installation* stage. When the DRM application is launched, the OS will obtain d_2 from the TPM to reconstruct d. With this improvement, in the unlikely event of a hacker breaking the TPM and retrieving d_2, he is unable to recover d directly, thus limiting the damage.
– *Dumping the DRM code from memory.* When a DRM application is loaded by an authorized loader, the DRM application will be stored in memory. A malicious user may attempt to dump the code for reverse-engineeering. This attack can be easily defeated when the OS insulates the process address space.
– *Forging decryption request.* In the enforcement stage, only the designed process can send the key request to decrypt the protected application, otherwise,

the malicious user can get the decrypted code with a debugger. To this end, the OS must verify the PCR of the loader such that only authorized loader can load all including os-encapsulated applications (*e.g.*, DRM application), but other loader such as debugger or browser can load non-authorized applications.

Since the application provider encrypts the DRM application, and only the encrypted application is stored in the non-protected storage such as hard disk. Therefore, the malicious user can not get the clear application code and can not start misuse attack. Therefore, neither can the traitor tamper/reverse-engineer the DRM application code, nor can he use a malicious debugger to load the code.

6 Conclusion

Since a TPM-enabled platform can prevent a user from running an "unapproved" application, it is possible to develop a tamper-resistant TPM-enabled DRM application, as suggested in [4, 17]. But it is not trivial to achieve that. We have presented two attacks on the basic TPM-enabled DRM: the abuse attack and the injection attack. The abuse attack misuses an "approved" loader (*e.g.*, debugger or browser) to run the DRM application, while the injection attack tampers with the genuine DRM application and re-signs the tampered version so that it can run as an "approved" application on the platform. Both attacks enables a traitor to gain unauthorized access to protected DRM content.

In order to counter these attacks, the present OS-encapsulation scheme makes sure that an encrypted DRM application is only loaded and decrypted by an authenticated OS, thus defeating the abuse attack, as well as the injection attack. This encapsulation format stops running virus-infected softwares, enables user to develop his own software, run debugger and browser but it still can enforce DRM protection.

References

1. Ravi Sandhu, and Xinwen Zhang,"Peer to Peer Access Control Architecture Using Trusted Computing Technology," the 10th ACM symposium on Access control models and technologies (SACMAT), pp.147-158, 2005.
2. Roger Schell, and Michael Thompson, "Platform Security: What is Lacking," Elsevier Science, Information Security Technical Report, Jan. 2000 `http://dx.doi.org/10.1016/S1363-4127(00)87628-1`
3. Trusted Computing Platform Allaince, "Tcpa main specification v. 1.2" `https://www.trustedcomputinggroup.org/specs/TSS/`.
4. Patrick George, "Smart Cards: A Bridge Between Users And Trusted Platforms," e-Smart Conference, 2004. see also `http://citeseer.ist.psu.edu/729848.html`
5. Vivek Haldar, Deepak Chandra, and Michael Franz, "Semantic Remote Attestation -A Virtual Machine directed approach to Trusted Computing," Virtual Machine Research and Technology Symposium, pp.29-41, 2004.

6. T. Garfinkel, M. Rosenblum, and D. Boneh , "Flexible OS support and applications for trusted computing," the 9th Hot Topics in Operating Systems (HOTOS-IX), pp.145-150, 2003.
7. E. Brickell, J. Camenisch, and L. Chen, "Direct anonymous attestation," 11th ACM Conference on Computer and Communications Security, pp. 132-145, 2004.
8. S.W. Smith, and D. Safford, "Practical Server Privacy Using Secure Coprocessors," *IBM Systems J.*, 40(3):683-695, 2001.
9. Alexander Iliev, and Sean W. Smith, "Protecting Client Privacy with Trusted Computing at the Server," *IEEE Security* & *Privacy*, 1(3):60-66, 2003.
10. J. Camenisch, "Better Privacy for Trusted Computing Platforms," 9th European Symposium On Research in Computer Security (ESORICS 2004), LNCS 3193, pp.73-88, 2004.
11. G.Wurster, P. C. van Oorschot, and A. Somayaji, "A Generic Attack on Checksumming-based Software Tamper Resistance," IEEE Symposium on Security and Privacy, pp.127-138, 2005.
12. P. C. van Oorschot, A. Somayaji, and G.Wurster, "Hardware assisted circumvention of self-hashing software tamper resistance," *IEEE Transactions on Dependable and Secure Computing*, 2(2):82-92, 2005.
13. H. Chang, and M. Atallah, "Protecting Software Code by Guards," Security and Privacy in Digital Rights Management, LNCS 2320, pp.160-175, 2001.
14. Bill Horne, Lesley R. Matheson, Casey Sheehan, and Robert Endre Tarjan, "Dynamic Self-Checking Techniques for Improved Tamper Resistance," Digital Rights Management, LNCS 2320, pp.141-159, 2001.
15. E.W. Felten, "Understanding trusted computing: will its benefits outweigh its drawbacks?," *IEEE Security* & Privacy, 1(3):60-62, 2003.
16. Ross Anderson, "'Trusted Computing' Frequently Asked Questions", `http://www.cl.cam.ac.uk/~rja14/tcpa-faq.html`
17. Bill Arbaugh, "Improving the TCPA specification", *IEEE Computer*, 35(8):77-79, Aug. 2002.
18. David Safford, "The Need for TCPA?," `http://www.research.ibm.com/gsal/tcpa/why\_tcpa.pdf`, Oct. 2002.
19. TCG Specification Architecture Overview, Specification Revision 1.2 28 April 2004.
20. OMA DRM Architecture, Approved Version 2.0 - 03 Mar 2006: OMAAD-DRM-V2_0-20060303-A `http://www.openmobilealliance.org/release_program/drm_v2_0.html`
21. OMA DRM Requirements, Approved Version 2.0- 03 Mar 2006: OMARD-DRM-V2_0-20060303-A `http://www.openmobilealliance.org/release_program/drm_v2_0.html`
22. OMA DRM Specification, Approved Version 2.0 - 03 Mar 2006 : OMATS-DRM-DRM-V2_0-20060303-A `http://www.openmobilealliance.org/release_program/drm_v2_0.html`
23. Pavol Cerven, "Crackproof Your Software," William Pollick publisher, 2002.
24. Hoeteck Wee, "On Obfuscating Point Functions," Annual ACM symposium on Theory of computing (STOC), pp. 523-532 2005.
25. Yongdong Wu, "Guarding Software Checkpoint", *International Journal Computer Scence and Network Security*, 5(12), 2005.
26. S. Chow, P. Eisen, H. Johnson, and P.C. van Oorschot, "A White-Box DES Implementation for DRM Applications," Digital Rights Management 2002, LNCS 2696, pp.1-15.
27. M. Jacob, D. Boneh, and E. Felten, "Attacking an Obfuscated Cipher by Injecting Faults," Digital Rights Management 2002, LNCS 2696, pp.16-31.