Available online at www.sciencedirect.com

ScienceDirect

Electronic Notes in Theoretical Computer Science

Electronic Notes in Theoretical Computer Science 197 (2008) 73–85

www.elsevier.com/locate/entcs

Implementing Trusted Terminals with a TPM and SITDRM*

Sid Stamm[a] Nicholas Paul Sheppard[b] Reihaneh Safavi-Naini[c]

[a] *Department of Computer Science, Indiana University, Lindley Hall, 150 S. Woodlawn Ave., Bloomington, IN 47405-7104, USA*

[b] *School of Computer Science and Software Engineering, University of Wollongong, NSW, 2522, Australia*

[c] *Department of Computer Science, University of Calgary, 2500 University Dr. NW, Calgary AB T2N 1N4, Canada*

Abstract

The SITDRM Enterprise system [1] protects private customer data by allowing customers to provide policies in the form of a machine-readable *license*. When employees of an organization want to use customers' data, they must be forced to abide by the licences provided. Some sort of hardened terminal must be used to ensure that not only the hardware and software will cooperate, but that the user of the terminal will too. We use the Trusted Computing Group's specifications for a trusted platform upon which to build a data user terminal that can be proved to implement correct license-enforcing behavior. A Trusted Platform Module (TPM) and a TPM-using operating system are all that may be required to construct a verifiably secure terminal.

Keywords: Trusted platform module. Digital rights management.

1 Introduction

Digital rights management (DRM) technology allows the use and dissemination of electronic information to be controlled by a machine-readable *license* that describes who may perform what action on the protected information, and under what conditions the action may be performed. DRM is well-known for its application to protecting copyrighted material, but can also be used to protect private personal information and sensitive corporate documents.

In order to restrict access to information according to licenses, DRM requires the existence of tamper-resistant terminals trusted to comply with any conditions imposed by licenses. Without the existence of such terminals, it is possible for

* This work was partly funded by the Smart Internet Co-operative Research Centre, Australia.

doi:10.1016/j.entcs.2007.10.015

an attacker to obtain access to information by constructing a phony terminal that simply ignores any conditions imposed by a license.

Software alone is not enough to verify that the users and terminal follow the rules set forth by the license. It is relatively easy for a technically-adept attacker to modify a software terminal in order to disable its licence-checking functions, or to extract sensitive information from it.

In this paper, we describe our experiences in adding support for hardware specified by the Trusted Computing Group [2] to a privacy protection system known as SITDRM Enterprise [1], which protects private customer data according to licenses supplied by customers.

We discuss how SITDRM and Trusted Computing (TC) fit together, and how one can use TC to implement such a terminal. We describe a prototype implementation of a secure terminal as well as difficulties encountered in development of the software. Finally, we describe how the prototype terminal was incorporated into the full SITDRM Enterprise system.

2 Related Work

Several TC platforms have appeared over the past decade, including Microsoft's dormant Next-Generation Secure Computing Base [6], the "Terra" trusted virtual machine developed by Garfinkel, et al. [5] and the "Enforcer" Linux Security Module developed by Marchesini, et al [4].

Marchesini, et al., in particular, describe a platform based on the Trusted Computing Group's specification, and discuss how their platform can be used to attest to the integrity of an off-the-shelf multimedia player, similar to the effect that we wished to achieve in our project. The player used in their demonstration, however, does not support DRM and they do not appear to have explored the development of a complete DRM system based on their platform.

3 SITDRM Enterprise

"SITDRM" (for "Smart Internet Technology DRM") is a DRM testbed developed at the Co-operative Research Centre for Smart Internet Technology in Australia. The system is based on the MPEG-21 Intellectual Property Management and Protection Components (ISO/IEC 21000-4:2006).

SITDRM forms the basis for a suite of applications known as "SITDRM Enterprise" that provide a privacy and document protection system for enterprises [1]. SITDRM Enterprise allows individuals to control the way in which their information is distributed and used by expressing their privacy preferences in a way enabling automatic enforcement by data users' computers.

SITDRM Enterprise applies the DRM model to information submitted via web forms, as shown in Figure 1. Data is created by a *data subject* (e.g. customer), and submitted in a protected form to a *data controller* (e.g. service provider). In order to access the protected data, a *data user* (e.g. employee) must obtain a license from

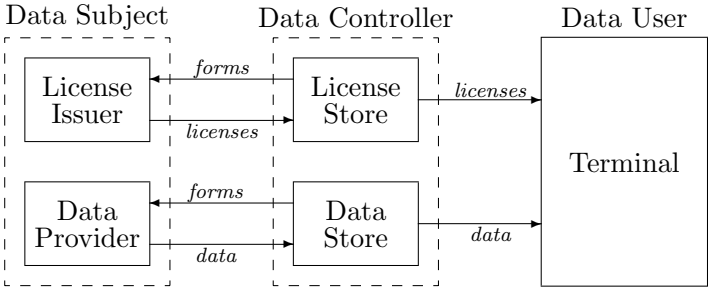

Fig. 1. Applying digital rights management to private data.

a *license issuer*. In this system, individual data subjects act as license issuers for their own data, and licenses grant the right to read data for the purpose that the data was uploaded.

The fundamental requirement for the system to be secure is that there exist tamper-resistant *terminals* trusted to uphold the conditions imposed by licenses, and whose compliance can be established by license issuers prior to any licenses being issued to them.

The design of such terminals and their integration into SITDRM Enterprise is the focus of the present paper. We present three goals as criteria for measuring the security of a data user terminal.

Goal 1: Prevent Key Exposure. Keys stored in memory should not be leaked to processes not complying with SITDRM licenses. When stored in main memory (unencrypted), keys can be leaked to third-party processes in many ways [7]. When the client software runs under a kernel that cannot be trusted to isolate one process' memory space from other processes (or does not securely deallocate memory), any keys stored in memory may be read by malicious third-party processes running on the same host. Leakage of these keys would allow a third-party process to masquerade as a legitimate SITDRM process.

Goal 2: Prevent Data Exposure. Data should not be leaked to processes not complying with SITDRM licenses. When being read or otherwise utilized, sensitive data must be kept unencrypted in memory. This memory must be managed in a secure manner similar to that for keys.

Additionally, data should not be transmitted to devices that may behave contrary to a corresponding license. If the behavior of an attached device cannot be verified, then the device could record unencrypted data and later use the data in violation of a license.

Goal 3: Verify Terminal Integrity. License issuers should have a way to determine that a terminal will enforce SITDRM licenses *before* sending it any licenses or encrypted digital items. Terminal integrity includes verification of the hardware and software composing the terminal—a valid configuration should somehow be proven to the data controller allowing the terminals to be maintained by someone other than the license issuer. As long as run-time verification can take place, the terminals can be purchased, installed and used without authorization of the

private data's subjects, or the manager of the license database.

If these goals are satisfied, every terminal T can be associated with a secret key SK_T and corresponding public key PK_T, such that the validity of PK_T can be attested to by the TPM that hosts the terminal. The secret key is known only by TPM and usable only by the terminal; in particular, it is not known to or usable by the human user of the terminal.

An item of data X can be protected by encrypting it with a unique *content key* Ck. Any license that grants a right to perform an action on that item contains Ck encrypted by the public key PK_T of the terminal on which that license is to be used. If license issuers only issue licenses to terminals that they have verified to be genuine as above, the content encryption key and therefore the data is accessible only to trusted terminals.

SITDRM assumes that every data user U is identified by a public key PK_U. In the implementation described in this paper, the corresponding private key SK_U is protected by the TPM as for terminal keys, but it is not used for any cryptographic purpose in licenses or protected data. In this way, the security of the system depends on the trustworthiness of terminals rather than the trustworthiness of the users, while users may still take advantage of the TPM's security features to protect their private keys.

4 Trusted Computing

"Trusted Computing" (TC) is used to describe a computing environment that can be trusted to behave appropriately — for example, the computer will not run malicious software. In the case of SITDRM, the system designers would trust a TC computer to correctly enforce any licenses issued.

The Trusted Computing Group (TCG) is a group developed in order to provide unbiased standards for TC. They have developed many specifications and standards for the Trusted Computing Platform, including functionality requirements and an API specification for software stacks to be implemented by hardware manufacturers [2].

The TC platform is composed of many pieces. Trusted Building Blocks are items that must be reasonably trusted on faith before the whole system can be trusted; they include elements such as a few CPU instructions used to initialize the system and RAM. The Trusted Platform Module (TPM) is a chip on the computer's main board and acts as the core engine of the TC system and it performs cryptographic operations, manages trusted data storage, and verifies the operation of various system pieces. With these elements and a TC certificate authority, a TC platform can be implemented providing the following features.

4.1 Measurement and Attestation.

A trusted computer needs to have the ability to prove that it should be trusted. This is done by "measuring" its current and previous operational states — including the

hardware and operating system — in a way that cannot be falsified. This involves extending a cryptographic hash chain and logging what was measured. If the boot process of a computer can be trusted (is measured as expected), then the currently running operating system can be trusted to behave correctly.

A TC *measurement* is simply a digest of some values. For example, a SHA-1 digest of a binary, embedded code, or hardware register values may be produced as a measurement for a computer's component or program. These measurements are stored in Platform Configuration Registers (PCRs). The order in which they are measured is enforced by *extending* the PCR by replacing the PCR's current value with $val_t \leftarrow SHA1(val_{t-1}|X)$, where X is the new measurement and "|" denotes concatenation.

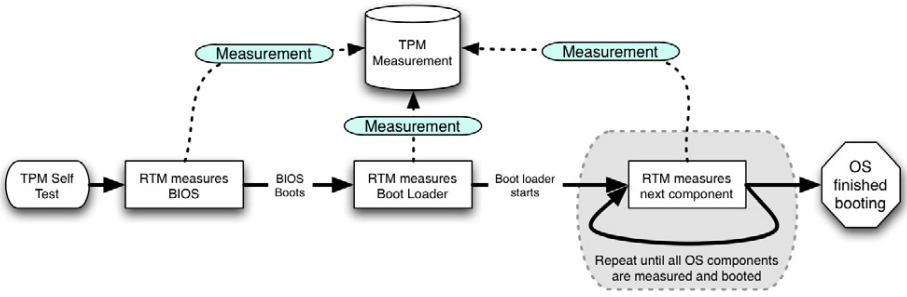

Fig. 2. How the TPM measures the boot process

When the computer boots, a *chain of trust* is formed by taking measurements of each part of the system as it starts, then extending a hash with the measured values (Figure 2). This trusted boot process is discussed in [2]. Applications can be measured by the OS before they are launched and the measured value is used to extend a PCR. Each measurement is also stored in a Stored Measurement Log (SML) which is used later during attestation for integrity verification.

When a third party wishes to verify the integrity of a terminal, it asks the terminal to attest to its integrity. In TC, this is sometimes referred to as *attestation*.

On request, the terminal sends the PCR and SML data to the challenger. The challenger uses the information recorded in the SML to recreate the PCR values. If the PCR values can be successfully recreated by the challenger, then the configuration of the terminal has been proven.

The challenger then analyzes the contents of the SML to decide whether or not the configuration is acceptable. To do this, the challenger must retrieve *credentials* issued by a trusted authority or the manufacturer of the software. These credentials contain information about the software (such as the digest and version), signed by a trusted authority, that are supposed to be used in the SML of a system using the subject of the credential.

4.2 Protected Cryptographic Operations.

The TPM also contains some protected cryptographic abilities. These are functions carried out in a trusted area of the computer, secluded from possible intrusion from

an untrusted OS or other malicious third-party software.

Key Generation and Storage. The TPM can be used for RSA key generation, protected storage, and encryption/decryption or signing/verifying operations. Key pairs can be generated by the TPM and then stored encrypted by the TPM's *storage root key* (SRK). The SRK is stored on the TPM itself in nonvolatile memory, and the private (decrypting) SRK cannot be copied from the chip. In this way, any key encrypted by the TPM can only be decrypted by the same TPM with the same SRK, and keys generated by the TPM and may be safely stored on the hard drive since the TPM is needed to "unlock" them.

Signing and Binding. Additionally, the TPM supports encryption (binding) and decryption (unbinding) as well as signing and verification using the RSA keys it generates. Keys can be generated an used on the TPM and without ever copying them in main memory; this avoids the risk of involuntary memory disclosure vulnerabilities. (For examples of these vulnerabilities see [7].)

5 A TC Secure Terminal

Our aim in using a TPM chip to harden a SITDRM data user terminal is to control as tightly as possible the existence and movement of secret keys and unencrypted data. Ideally, we would like to be able to use the TPM as a trusted input/output channel between the SITDRM software and output devices. Unfortunately this is unrealistic, since the TPM is not designed to be used as a cryptographic processor. We consider a situation where a custom SITDRM data user terminal is run on a trusted platform.

5.1 Ideal Secure Terminal

An ideal secure terminal would ensure that the data and keys encrypting the data are only used correctly and by the correct terminal users. Such a terminal would be considered "trusted" in the fashion that a TC device is trusted. This simply means that the computer, its attached devices, and its user will all be restricted to usage allowed by SITDRM licenses.

- **Platform Security.** The operating system and hardware comprising the terminal's platform would be trusted if the SITDRM software could run on the terminal with complete memory protection. Additionally, the kernel would need to relinquish all memory ownership rights to the SITDRM process while it was running, giving the SITDRM software exclusive control over its memory to hide keys and data from third-party programs. A terminal is deemed *platform secure* if the operating system and hardware behave in a way that cannot breach the SITDRM licenses, thus prohibiting a third-party program from "sniffing" the memory used by the SITDRM software.
- **Device Security.** All devices attached to the terminal must also be bound to the SITDRM licenses. This means that if a license allows viewing but not recording, the device used for viewing must prohibit recording of the data. The devices

should be provably conforming to the SITDRM licenses; they cannot be modified to have different capabilities without being approved by the SITDRM software.

- **Communication Security.** As data is transferred in and out of the terminal (as well as to output devices on the terminal), it should not be vulnerable to eavesdropping. Transmissions over "sniffable" lines should be encrypted to prevent misuse or copying of decrypted data. For example, data should be encrypted as it passes through the cable between the video card and the monitor since a recording device could be inserted between the two components.

5.2 Our Terminal

In this section, we describe the architecture of a tamper-resistant terminal based on the TCG's specifications and the SITDRM platform. We have simplified some of the aspects of the full SITDRM Enterprise system in order to facilitate a straightforward illustration of the techniques employed. We will discuss integration of the simplified terminal into the full system in Section 6.2.

5.2.1 Operation

The data user software (which we refer to as the software from here on) is launched by the operating system. Before execution of the software is allowed, the code is measured and a PCR is extended with the measurement, logging the launch event in the TPM's SML (Figure 3). The software then begins executing by loading the *terminal key* into the TPM (since it is probably stored — encrypted by the TPM's storage key — somewhere outside the chip) and prompts the user for a password. The user's password is used to unlock the *user key* and also load that into the TPM.

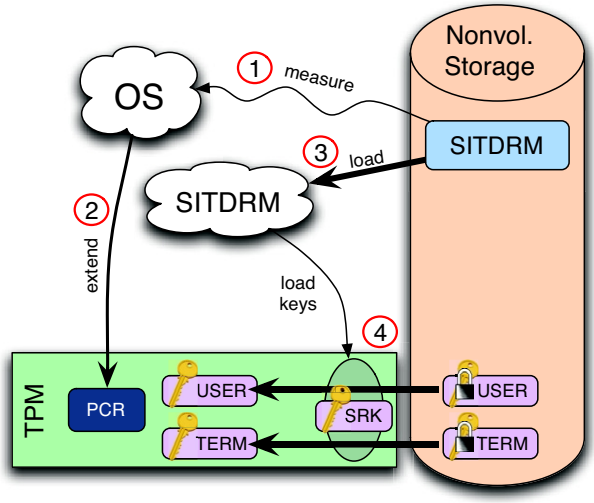

Fig. 3. When launching, (1) the data user software is measured, (2) the PCR value is extended, (3) the data user software begins executing, and then (4) the user and terminal keys are loaded into the TPM.

Once launched, the software permits the data user to select items of data that are available for viewing. When an item is selected and the user requests to perform

some action (such as view, email, or print), the software attempts to locate a license allowing it to decrypt the data. If it can find one, the encrypted content key is sent to the TPM for decryption (Figure 4). The decrypted content key is returned and the cryptographic engine in the data user software decrypts the digital item.

Once the requested action is complete, the content key and decrypted data are erased from memory. This ensures that the key and unencrypted data are not present longer than necessary, minimizing the possibility of attack. Memory shredding techniques [8] can be used to help decrease the chance of a vulnerability.

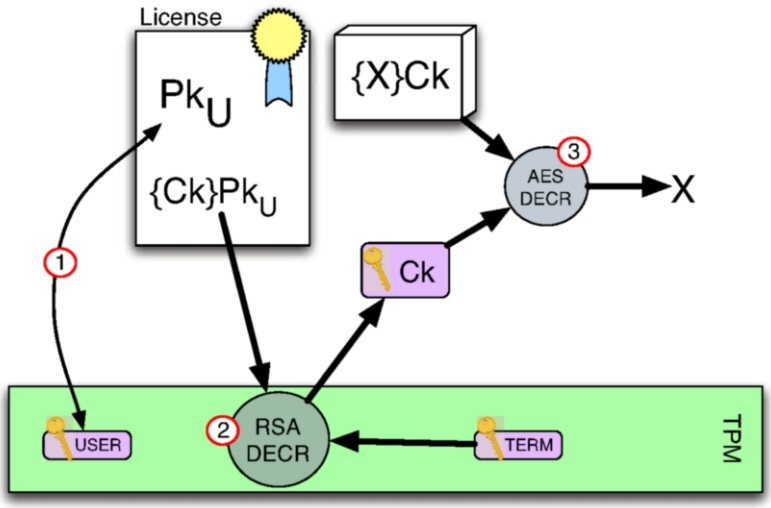

Fig. 4. The steps taken to decrypt a digital item in the demonstrator: (1) verify the user's public key, (2) extract the resource key, (3) decrypt the digital item.

If a license to perform an action is not already installed on the terminal, the terminal can request one from the licence issuer responsible for the data. Before issuing any licenses, the license issuer must verify that the terminal is a trusted one (in addition to satisfying any other conditions it might have for issuing licenses).

This verification can be done by adding an Authentication Server (AS). Like granting tickets in the Kerberos5 system [9], the AS could check that a terminal is valid, then let the license issuer and data repository know that this user/terminal combination is allowed to download digital items and licenses.

Authentication to the AS involves proving the integrity of the data user software. A secure SITDRM Enterprise system requires that data and licenses are only issued and distributed to terminals that will abide by the restrictions outlined in the licenses. Authentication must be established with the following method. Failure of any of the steps terminates the connection to the server in order to prevent disclosure of licenses or encrypted digital items to an attacker.

(i) User ID presented to the server

(ii) User and terminal keys presented to the server, encrypted with the server's public key

(iii) TC integrity data presented to the server (PCR, SML, and credentials)

Once the PCR hash has been validated and the SML is accepted by the AS, the data user client is allowed to request digital items and licenses from their respective sources. This data should be transferred via a secure channel.

5.2.2 Security Analysis of this Design

In order to satisfy the three goals specified in Section 3, we must make a few basic assumptions:

Software-only Adversary. Hardware attacks will not be performed on signals transmitted between components of the computer. An adversary will most likely have access to the software in the computer, so a software-driven attack is possible.

Secure Memory. The memory space for the data user software will be secure and isolated from other processes. This is a safe assumption, since on a trusted platform, the operating system is part of the chain of trust (Section 4). A trusted operating system should disallow applications that it runs from accessing each other's memory space. It should also provide mechanisms to help prevent data leaks from unallocated memory or virtual swap space.

Based on the description of a secure terminal presented in Section 3, and the adaptation described in Section 5.2, a secure TC-based terminal can be constructed.

Goal 1: Prevents Key Exposure — There are three kinds of keys to be considered: the RSA private key for the user, the RSA private key for the terminal, and the symmetric keys for content access. The RSA keys are generated by the TPM and are encrypted by the TPM's SRK so they can only be used by the TPM. The software's memory is isolated from other processes by the operating system; additionally, the content keys are decrypted as needed, then deleted from memory. In this way, resource keys are stored decrypted in main memory only while they are needed. They are never stored decrypted on persistent storage.

Goal 2: Prevents Data Exposure — As described above, the software's memory is isolated from other processes by the operating system. Additionally, data is decrypted as needed for an action, then deleted from memory. Data is never stored decrypted on persistent storage.

Goal 3: Verifies Terminal Integrity — By following the TC integrity verification process, the license issuer is able to test if a terminal is going to follow the rules specified by SITDRM licenses. Changes in the SITDRM data user software are easily detected as an unexpected value in the SML, and untrustworthy system configurations can be detected as well.

6 Implementation

We constructed a prototype client using the TPM Software Stack (TSS) [3]. The purpose of the demonstrator was to exhibit the ability to adapt the current SITDRM data user client (known as "IPDoc") to a TC platform as described in Section 5. As noted earlier, some aspects of the SITDRM system were simplified in order to make

the initial prototype easier to understand. Integration of the simplified prototype into the full SITDRM Enterprise system is dicussed at the end of this section.

6.1 Design

There were three elements created in development of this demonstrator: a license and data server, network protocol, and data user client. The only piece of the system that used a TPM was the client; the other two components were developed as support software for the client.

6.1.1 License and Data Server

For simplicity, we combined the AS, license issuer, and data repository into one server that authenticates a client then serves licenses and data. Acting as the license issuer, a data repository, and a user manager, the server was able to authenticate a client connection (by validating users' credentials against a list) and then create/issue licenses and serve encrypted data. As well, the server had the ability to perform integrity verification.

Digital Items (DIs). The server provided a simple interface to whomever runs the software; it allowed an administrator to import arbitrary data files which were then encoded into Base64, then encrypted and encapsulated in an XML format specified by MPEG-21's Digital Item Definition Language. These items were encrypted with a unique secret content key, and stored on the server's hard disk. Content keys were generated using a one-way function of the server's secret master key and the item's identifier, so it was not necessary to store them individually.

Licenses. The server provided the ability to issue licenses. In this demonstrator, all users in the system who requested a license were granted a simple license with the "play" right for all digital items. In a real system, of course, licenses would only be granted if the requester satisfied some policy of the license issuer.

The license was generated and issued using the existing SITDRM software API, a license authority RSA key pair, and the server's master key. These licenses contained a reference to the DI, the DI content key (encrypted with the grantee's public key), and the "play" right. Licenses are signed by the license manager to prevent tampering, and are stored on the server's hard drive until they are requested by the grantee of the license.

Authentication. The server also implemented authentication as required to prove a user's identity, a terminal's identity, and a terminal's integrity. This authentication is basically user-lookup and key verification, then verification of SML and PCR confirmation in the TC style.

6.1.2 Data User Client

The client software provided a simple interface that allowed the user to connect to a server, download licenses or digital items, then exercise grants provided by any downloaded licenses. The client implemented a client-side of the network protocol when communicating with the server. It also made heavy use of the TPM.

Self-Measurement. Since the OS used for development was not TC-enabled, our program had to create the measurement events itself. This was done by extending the PCR with the SHA-1 hash of the executing code.

Key Generation and Storage. The RSA keys for the user and terminal were generated by the TPM on request. As a simplification, the terminal and user key were combined into one — since the license model was simplified, there was no need for two RSA key pairs.

Decryption. Content keys in licenses were sent to the TPM for decryption, since that is where the user private key was housed. In this way, the user key never left the TPM unless encrypted by the TPM's SRK.

Licenses were parsed and enforced using the SITDRM API. Each grant present in the license were represented by a button that when pressed allowed the user to exercise it. This was only a single "play" grant for licenses issued by the demonstrator server. When pressed, the "play" grant exercise button opened a new window and rendered the document. The window launched a new rendering thread and plugged into the SITDRM API's IPMP engine to temporarily decrypt the document.

Though this prototype illustrates the majority of the TPM implementation, secure memory was not established. The development platform contained only an emulator for the TPM chip, and so the OS was not a trusted operating system. Additionally, there was no support for isolating process memory spaces on the development system, so memory disclosure attack vulnerabilities may still have been present.

6.2 Integration into SITDRM Enterprise

In the simplified scenario described above, it is necessary for the license issuer to know the public key of the terminal on which a digital item is to be used before a license to use that item can be issued. This is obviously impractical in the scenario shown in Fig. 1: data subjects have no way of knowing which terminals inside an organization will be used to process their data.

SITDRM Enterprise solves this problem by using a simple role-based access control model. In addition to the key pairs possessed by terminals, SITDRM Enterprise assumes that every role R is associated with a key pair SK_R and PK_R. We assume that the public keys of all users and roles can be verified using a public key infrastructure.

Role-based access control is implemented using a two-stage licensing process: *membership certificates* permit individual data users to act as members of roles using the `PossessProperty` right, while *resource licenses* permit members of roles to perform actions using the `PropertyPossessor` principal. In order for a particular data user to carry out an action on a document, he or she must obtain both a resource license that permits some role to carry out that action, and a membership certificate that makes him or her a member of that role.

A membership certificate identifies a member of the role by his or her public key, and contains the private key of the role, encrypted by the public key of the

terminal on which that membership certificate is to be used.

Membership certificates can be obtained from a *role issuer* operated by the data controller. The role issuer takes on the role of the authentication server described earlier, and will only award membership certificates to terminals that it has established can be trusted by TC's integrity verification process. Note that the data subject must trust the role issuer to manage role memberships in a responsible fashion.

Resource licenses are issued by data subjects to roles and contain the content key encrypted by the public key of the role to which the license has been issued. If a terminal establishes that it has the right to perform an action on the digital item, it may decrypt the content key from the resource license using the role's private key in the membership certificate.

7 Discussion

Implementing a client that uses the TPM for cryptography, key management, and integrity reporting is not a simple task. The Trusted Computing Group has published an enormous specification for each level of their architecture including the TSS core services — the level used by the demonstrator. As a result, some portions of the hardened terminal were non-trivial to implement.

Before any functionality of the TPM can be used, its "take ownership" process must take place. This involves creating some identifiers inside the TPM as well as a Storage Root Key (SRK). Once these are generated, the TPM can be used. Additionally, the TPM emulator kernel module must be loaded into the kernel and initialized before the TrouSerS daemon will allow API access to the TPM.

TPM keys are created with a specified type, either as a signing key, binding key (for encryption) or as a legacy key (general use). After some testing, it was determined that the signing and binding keys do not always work when the public key is exported and used with different software. Additionally, when the key is created, its encoding method must be specified as either RSA v1.5 or OAEP. Omitting one of these specifications when creating a key causes the TPM to assign defaults which are not specified by the TCG and can result in unexpected behavior.

The TPM can use either RSA v1.5 or OAEP encoding methods for encrypted data. OAEP encoding allows a pseudonym to be inserted into padding, which the TPM uses to insert the string "TCPA". Unfortunately the Java Cryptography Extensions do not easily support the pseudonym feature, so it is not possible to use the TPM with OAEP padding from Java and we had to to use RSA v1.5 encoding even though OAEP is the encoding method recommended by the XML security specifications.

Due to the nature of TC integrity reporting, it is difficult to properly perform integrity verification. First, each item recorded in the SML must be issued a credential; for example, the OS must have a credential issued by the manufacturer and the video card must have a signed driver. Second, the credentials must be signed and verifiable using some public key infrastructure — much like SSL certificates. These

credentials do not exist (are not created by the manufacturers) and they are not distributed in a PKI fashion. If a public key infrastructure were implemented properly to distribute credential signer certificates, then integrity reporting is strong. Until then, a third party wishing to verify my terminal is trusted must simply trust on blind faith all certificates we provide.

8 Conclusions

We have shown how the TC platform can be used to harden a DRM terminal in a way that provides for license-abiding use of protected content. Requirements for this to work properly have been outlined as being: a trusted operating system, a TPM chip, and an authentication server.

TPM chips are now included in many new commuters, and the most recent versions of both the Windows and Linux operating systems contain at least some support for TPMs. TPMs are therefore a very attractive option for securing DRM terminals.

References

[1] N. Sheppard, R. Safavi-Naini "Protecting Privacy with the MPEG-21 IPMP Framework". International Workshop on Privacy Enhancing Technologies 2006, pp. 152-171.

[2] Trusted Computing Group, "TCG Specification Architecture Overview." Revision 1.2, 28 April 2004. https://www.trustedcomputinggroup.org/groups/TCG_1_0_Architecture_Overview.pdf

[3] Trusted Computing Group, "TCG Software Stack Specification." Version 1.2. https://www.trustedcomputinggroup.org/specs/TSS_1_2_Errata_A-final.pdf

[4] J. Marchesini, S. W. Smith, O. Wild, J. Stabiner and A. Barsamian, "Open-Source Applications of TCPA Hardware". Annual Computer Security Applications Conference 2004.

[5] T. Garfinkel, B. Pfaff, J. Chow, M. Rosenblum and D. Boneh, "Terra: a Virtual Machine-Based Platform for Trusted Computing". ACM SIGOPS Operating Systems Review 37(5):193-206.

[6] Microsoft Corporation. "Next-Generation Secure Computing Base", http://www.microsoft.com/resources/ngscb/default.mspx

[7] G. Di Crescenzo, N. Ferguson, R. Impagliazzo, M. Jakobsson. "How to forget a Secret," In C. Meinel and S. Tison, editors, Proceedings of STACS 1999, volume 1563 of Lecture Notes in Computer Science, pages 500-509. Springer, 1999.

[8] J. Chow, B. Pfaff, T. Garfinkel, M. Rosenblum. "Shredding Your Garbage: Reducing Data Lifetime Through Secure Deallocation," In Proc. 14th USENIX Security Symposium. August, 2005.

[9] B.C. Neuman and T. Ts'o. "Kerberos: An Authentication Service for Computer Networks," IEEE Communications, 32(9):33-38. September 1994.

[10] M. Strasser, "A Software-based TPM Emulator for Linux," Semester Thesis, Department of Computer Science, Swiss Federal Institute of Technology Zurich. Summer 2004. http://www.infsec.ethz.ch/people/psevinc/TPMEmulatorReport.pdf