I.J. Wireless and Microwave Technologies **2012, 4, 1-7**
Published Online August 2012 in MECS (http://www.mecs-press.net)
DOI: 10.5815/ijwmt.2012. 04.01

Available online at http://www.mecs-press.net/ijwmt

Securing Peer-to-Peer Distributions with Trusted Platform Modules

Hao Li, Yu Qin, Qianying Zhang, Shijun Zhao

State Key Laboratory of Information Security, Institute of Software Chinese Academy of Sciences, Beijing, China

Abstract

We present a novel **solution** that allows one platform to securely distribute or redistribute digital contents to another in P2P networks. The solution is based on platforms with Trusted Platform Modules (TPMs). It maintains the confidentiality and freshness of digital contents during the processes of distribution. Given an ideal (tamper-proof) trusted platform, the solution can even withstand attacks by dishonest users during the processes of usage. Moreover, it can also be used to redistribute n-time-use digital content offline, so it is more flexible and scalable than other related distribution solutions to enable widespread deployment. Lastly, by adding a few simple features to TPMs, our solution can easily prevent the malicious sender and receiver from colluding when the redistribution takes place, so we can ensure that they can not gain more than a previously defined amount of rights without contacting the content provider.

Index Terms: Trusted Computing, TPM, peer-to-peer, redistribution, n-time-use digital content.

1. Introduction

In the traditional client-server architectures, there is always a trusted server and a client that connects to the server to acquire certain contents. The contents in the server can be protected by various effective security mechanisms, but it is difficult to protect them when they are beyond the control of a server. Since clients are often devices that are logically and physically under the control of their owners, client users can attack and circumvent the protection mechanisms easily. It will be more complex in the P2P distribution architectures because the party can be both client and server at the same time. That is, all the interests of different parties should be reflected in the P2P architectures.

Fortunately, the Trusted Computing Group (TCG) has specified a Trusted Platform Module (TPM) acting as a trusted third party which can be used to build trust relationships between users in the P2P networks. Nowadays, TPMs have been embedded in many personal computers. So we can get an ideal trusted platform based on such a chip. And the technologies of building such platforms have been focused on for several years, such as [1-6]. In this paper, we need such trusted platforms to provide secure environments in which our solution runs. However, how to build such a platform is beyond the scope of this paper (the reader can get more

Corresponding author:
E-mail address: lihao@is.iscas.ac.cn

about how we build a trusted platform in [3, 5]).

Contribution. We present a solution for offline, peer-to-peer content sharing which allows redistribution of n-time-use content. The basic principle is to use TPM migratable keys with transport session logs (acting, in essence, as use-count certificates) in order to prevent replay and a man-in-the-middle style attack. A further process is described for preventing collusion by two parties in the P2P networks which need a modification to the TPM. Finally, we give an informal analysis of our solution's security, and the results of performance experiments.

2. Related Work

Securing P2P distribution using trusted computing has already been introduced by [7] for several years. And some concrete schemes of distribution have been proposed based on different models and assumptions [4,8,9,10].

In [8], Sandhu and Zhang present an architecture that provides access control using a trusted hardware component such as a TPM, a secure kernel, sealed storage, and a trusted reference monitor that interacts with applications through secure channel. However, the secure distribution is just described in a high-level, and replay attacks are neglected. In [4], Kyle and Brustoloni implement a novel Linux Security Module (UCLinux) . In order to distribute secrets, UCLinux just uses a version of TLS v1.0 protocol, extended to include attestation during its handshake. However the distributions of digital contents are not discussed and the security analysis of the solution is not presented.

The authors of [9] present a protocol that allows servers to securely distribute secrets to trusted platforms. However the distribution model in [9] is based on traditional client-server architectures. So it can not support offline redistributions in P2P architectures.

In [10], the authors identify characteristic P2P scenarios and demonstrate how distributions between peers can be realized by applying a few basic licensing operations. Then, they present a security architecture to realize these basic license operations. However, the distributions of n-time-use digital contents are not discussed, which are common scenarios in P2P architectures.

3. Security Requirements

Consider a user, Bob, who downloads a song from a content provider called Alice. Bob has paid for the song, and he can only use it for 10 times. Unfortunately, Bob will not follow the rules voluntarily because he has different interests. This is a typical scenario which usage control has focused on for several years. As we just focus on the processes of (re)distributing digital contents in this paper, we suppose that there is a usage-control system in each peer's platform and Alice can define the values of platform configuration registers (PCRs) in a TPM which reflect the usage-control system.

Our setting is shown in Figure 1. Alice provides the song *s* that it will be downloaded to untrusted storage in Bob's platform over an open channel. And later Bob wants to share *s* with Dave, so he redistributes it to Dave over another open channel. Hence, our solution should protect *s* from the man-in-middle attacks (as shown in Figure2). Furthermore, Bob may not respect the rules of Alice (as shown in Figure 2, Bob may be dishonest). In any case, Alice is willing to distribute *s* to Bob's platform which is known to meet her security requirements (i.e., if Bob's platform meets Alice's security requirements, it will allow Bob to use *s* following Alice's rules).

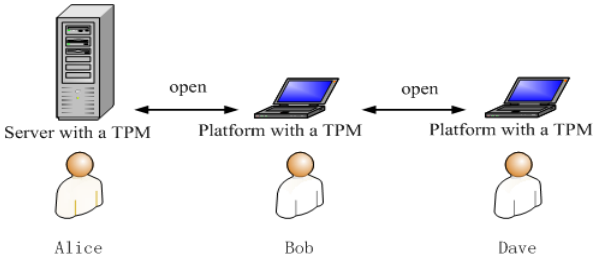

Figure 1. Setting

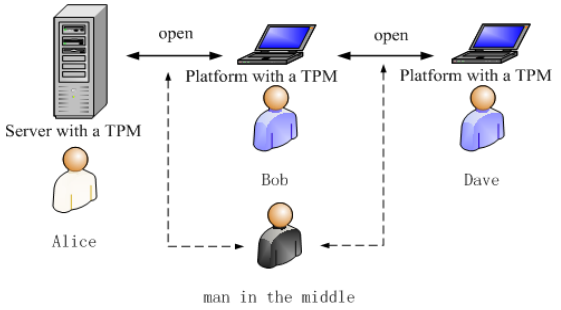

Figure 2. Attacks

We consider that Bob has listened to *s* for 5 times, and then he redistributes it to Dave. Thereby, Dave can still listen to *s* for 5 times. Our solution should protect the interests of them (i.e., the 5 use times of *s* transferred form Bob to Dave should be acknowledged by both of them). In addition, Bob and Dave may collude in order to get more rights lawlessly as shown in Figure 2. So our solution should prevent them from colluding.

From the setting and possible attacks described above, we identify the following security requirements:

1. Trust verification of peers' platforms, so that the digital content provider can believe that his or her interests can be insured.
2. Confidentiality of digital contents in transit between and in storage on the peers' platforms, so that unauthorized reading can be prevented.
3. Freshness of n-time-use digital contents in transit between and in storage on the peers' platforms, so that replay attacks can be prevented and therefore peers can not get more use times than they are permitted.

4. Solution

4. 1 Trust Verification of Platforms and Confidentiality of Contents

The protected storage feature of a TPM allows for the secure storage of TPM keys, as described in [11,12]. We can only use the private key created by a TPM when the current environment is the one stated while creating it. Therefore we can provide confidentiality of the digital contents, and verify the state of peers' platforms by encrypting the contents with the private key created by a TPM (sometimes, the content is encrypted by a symmetric key, which is protected by the private key in a TPM). As shown in [9], the distribution method based on protection of private keys has already existed. However, this method needs the content provider always online to verify that the binding key created by receiver is a non-migratable key and it is sealed to a set of PCRs required by the provider. For our setting in which offline redistributions usually take

place between peers, this method is not appropriate. So we use another approach founded upon both the TCG key migration [12] and protection of private keys, as shown in Figure 3.

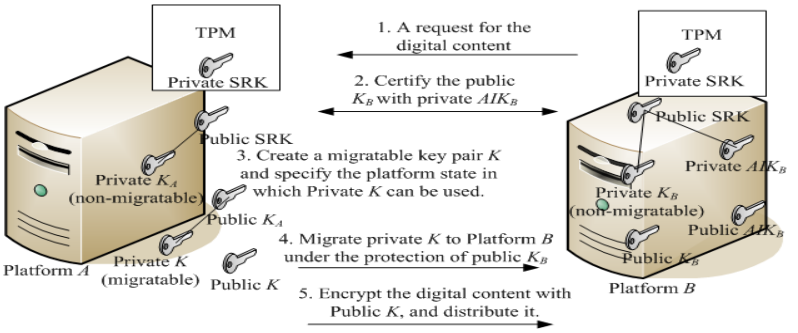

Figure 3. An approach for the digital content (re)distribution with TPM

Figure 3 illustrates the process of our approach. In the first step, Platform *B* sends a request for the digital content to Platform *A*. And in the second step, Platform *B* generates a non-migratable key pair K_B, and certifies public K_B using its private AIK_B. By verifying the AIK-certified public K_B, Platform *A* ensures that K_B is a non-migratable key. And then, Platform *A* generates a key pair *K* which is used to encrypt the digital content. And the state of platforms in which *K* can be used is specified at the same time (by specifying the PCR values). Therefore, Platform *B* can use *K* to decrypt the digital content only if its state matches that bound to *K*. After that, *K* is migrated from Platform *A* to Platform *B* under the protection of public K_B (for more details, the reader is referred to TCG key migration [12]). Finally, Platform *A* encrypts the digital content with public *K*, and sends it to Platform *B*. Because of the protection of private keys provided by TPMs, this approach can prevent the dishonest user and man in the middle from getting the digital content. And moreover, the security state of platforms in which the content can be decrypted is specified by the content provider. So when redistributions take place between peers, the Step 3 can be omitted (i.e., redistributions can take place between peers without contacting the content provider).

4. 2 Freshness of n-time-use Digital Content

Considering the P2P networks, we can not provide an online server to record the most recent version of the digital content (e.g., remaining use times). So it can only be recorded in the untrusted storage of the user's platform. That is, what we need is some form of trusted memory on the machine that is somehow changed irreversibly during usage processes, such that it would be infeasible for attacker to revert the machine to a previous state. Therefore in this paper, we present an approach based on the TPM monotonic counters, which is a kind of irreversible memory provided by TPM, in order to guarantee the freshness of n-time-use digital contents. However, there is a limit to the number of physical counters in TPMs. So we have to use the transport sessions of TPM to bind lots of digital contents to one physical counter.

As shown in Figure 4, we suppose that there is already a physical counter in the TPM, which is called $Counter_A$. Then, we establish a transport session, and execute the increment of $Counter_A$, which returns its current value Value($Counter_A$) after increment. At last we execute TPM_ReleaseTransportSigned in order to get the log. Note that, we replace its input parameter *AntiReplay* with a hash value Hash(public *K*|*flag*|*nonce*). Public *K* is the encryption key of a digital content, and *nonce* is a random number. And the *flag* can be "created", "used" or "migrated", which indicates that Value($Counter_A$) is bound to the creation, use or migration operation of *K* respectively. Therefore, the transport session log includes $Counter_A$, Value($Counter_A$), encryption key of the digital content, *flag* and a random nonce *nonce*, and then the association between digital contents and TPM counter values can be built directly. We denote this log with $\text{TransLog}_{\text{AIK}}$ ($Counter_A$,

Value($Counter_A$), public K, $flag$, $nonce$). If we find a change of the counter's value without a transport session log, we consider it as a replay attack. Hence, offline (re)distributions of n-time-use digital contents can be built with these transport session logs.

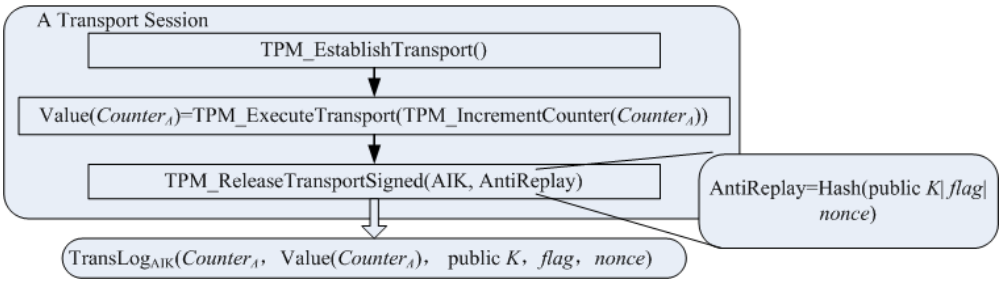

Figure 4. An approach for the freshness protection of n-time-use digital content

4. 3 Anti-collusion Approach

In P2P scenarios, two dishonest users may collude in order to get more use times. For example, Dave may not verify Bob's transport session logs.

We propose an approach to prevent dishonest users from colluding by adding a few simple features to the TPM. TPM specification 1.2 has provided two key migration approaches, but none of them executes the verification of platform state when migration takes place [12]. So we add two sets of PCRs when generating a key with TPM_CreateWrapKey. The first one is called Migration PCRs, which specifies the state of the source platform. Then the key can be migrated only if the state of the source platform matches Migration PCRs. The other one is called Conversion PCRs, which specifies the state of the destination platform. Then the migrated key can be converted to the TCG storage form only if the state of the destination platform matches Conversion PCRs. Alice can specify the two new sets of PCRs when she create K in order to ensure that there is a security software which executes all the steps of redistribution which include verifying of these session logs. And if Dave does not verify the logs, the PCR values of his platform would not match PCR_{Req} specified by Alice. Hence he cannot decrypt the migrated key. Then he cannot get the digital content. Therefore our solution can prevent Bob and Dave from colluding by adding the new feature.

5. Security Analysis

Since the communication channels are all open between peers, a man in the middle can get all the messages exchanged in the distribution and redistribution processes. However, the digital content is encrypted by Alice using public K. And the private K is protected by TPM. Moreover, K is protected by the public key of a non-migratable key from destination platform, when it is migrated to other platforms. And the non-migratable key is also protected by the TPM of the destination platform, which can be certified to the sender's platform. Hence, the man in the middle can only get the digital content or private K by breaking the cryptographic system, which we assume to be infeasible. Thus our solution is secure against a passive man-in-the-middle attack. And the active man in middle may replace the messages exchanged in the open channel. There are two kinds of messages which can be replaced. The first kind of messages is the certifications of keys or the transport session logs. These messages are all signed with AIKs, and AIKs are all protected by TPMs. So the man in the middle can not fool the peers because he can not forge the signatures. The other kind of messages is the encrypted migration blob of a key or the encrypted digital content. The replacement of this kind of messages amounts to the Denial of Service attacks, which is beyond our scope of this paper.

After the digital content reaches the destination platforms, the dishonest user may want to use the content

without following the provider's rules. However, he could put his platform into a dishonest state (by launching another process or rebooting another stack) which results in different PCR values. Hence, he can not decrypt the content using private *K* in the dishonest state of his platform. And if the platform state is honest, it will execute all the steps of usage in order to get the content. Then the replay attacks can be detected by the verification of transport session logs.

Finally, we add two sets of PCRs when generating a key with TPM_CreateWrapKey as described in the former section. The content provider can specify two new sets of PCRs when he creates the encryption key in order to ensure that the states of the source and destination platforms are trusted when the migration takes place. The dishonest peers could redistribute contents without following the right steps of redistribution, which will result in different PCR values. Hence, they can not generate a migration blob of the encryption key or convert it into the TCG storage form. Thus our solution can prevent the malicious peers from colluding.

6. Performance Analysis

We do the performance tests on a PC, with a 2.00 GHZ Pentium 4, and 1 GB main memory. The TPM chip is from National Semiconductor Corporation. Figure 5 shows the experimental performance results of key operations in our solution.

Although the performance of our solution may not be good enough, we believe that it is acceptable in the P2P distribution architectures because there is not an online server which has to execute lots of these operations in a short time. All these operations are executed in the respective peers' platforms. For example, the cost of several seconds is acceptable for a peer who wants to get a large movie file by running our solution. So our solution is usable in P2P distribution architectures.

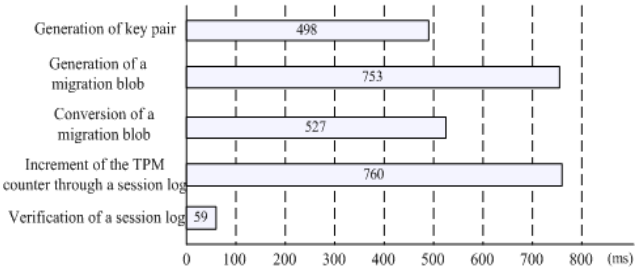

Figure 5. Performance Measurements

7. Conclusion

In this paper, we present a TPM-based solution that allows one platform to securely distribute or redistribute digital contents to another. We identify characteristic P2P distribution settings and list the security requirements which our solution should satisfy. After that we describe our solution in details. That is, we explain why our solution can support the offline redistribution of n-time-use digital contents, and how it does. Moreover, we propose some changes to the TPM that can make our distribution solution simply prevent peers from colluding. Then we provide an informal analysis of the security of our solution. Finally, we present the performance results of some key operations, which can be acceptable in P2P architectures.

Acknowledgment

This paper is supported by the National Key Technology R&D Program of China (2008BAH22B06), and the CAS Innovation Program (ISCAS2009-DR14, ISCAS2009-GR03).

References

[1] Sailer R, Zhange XL, Jaeger T, and Doorn LV. Design and implementation of a TCG-based integrity measurement architecture. //Proceedings of the 13th USENIX Security Symposium, San Diego, 2004. San Diego: USENIX Security Symposium, 2004: 223-238

[2] Alam M, Seifert MP, Li Q, Zhang XW. Usage control platformization via trustworthy SELinux. //Proc. of the 2008 ACM symposium on Information, computer and communications security (ASIACCS), Tokyo, 2008. Tokyo: ACM Press, 2008: 245-248.

[3] X. Chu and Y. Qin. A Distributed Usage Control System Based on Trusted Computing. In Proc. of 1st Trust Computing Theory and Practice Conference, 2009.

[4] D. S. Kyle and J. C. Brustoloni. UCLinux: a Linux Security Module for Trusted-Computing-based Usage Controls Enforcement. In Proc. of 2nd ACM Workshop on Scalable Trusted Computing, 2007.

[5] Li Hao and Hu Hao. UCFS: Building a Usage Controlled File System with a Trusted Platform Module. In Proc. of 1st Trust Computing Theory and Practice Conference, 2009.

[6] X. Zhang and J.-P. Seifert. Security Enforcement Model for Distributed Usage Control. In IEEE International Conference on Sensor Networks, 2008.

[7] S. E. Schechter, R. A. Greenstadt, and M. D. Smith, Trusted Computing, Peer-To-Peer Distribution, and the Economics of Pirated Entertainment, The Second Annual Workshop on Economics and Information Security (EIS'03). College Park, Maryland, May 29-30, 2003.

[8] R. Sandhu and X. Zhang, Peer-to-Peer Access Control Architecture Using Trusted Computing Technology. In: SACMAT 2005, Stockholm, Sweden (June 2005)

[9] P. E. Sevinc, M. Strasser, and D. Basin. Securing the distribution and storage of secrets with trusted platform modules. In WISTP 2007, pages 53–66, 2007.

[10] A. Osterhues, A. R. Sadeghi, M. Wolf, C. Stuble, and N. Asokan. Securing Peer-to-peer Distributions for Mobile Devices. In 4th Information Security Practice and Experience Conference, 2008.

[11] Trusted Computing Group: TCG architecture overview. (TCG Specification)

[12] Trusted Computing Group: TCG TPM specification version 1.2. (TCG Specification)