I.J. Computer Network and Information Security, **2011, 2, 26-33**
Published Online March 2011 in MECS (http://www.mecs-press.org/)

Securing the Distributions in P2P Networks with Trusted Platform Modules

Hao Li
State Key Laboratory of Information Security/Institute of Software/Chinese Academy of Sciences, Beijing, China
Email: lihao@is.iscas.ac.cn

Yu Qin, Qianying Zhang, and Shijun Zhao
State Key Laboratory of Information Security/Institute of Software/Chinese Academy of Sciences, Beijing, China
Email: {qin_yu, zhangqy,zhaosj}@is.iscas.ac.cn

Abstract—**We present a novel solution that allows one platform to securely distribute or redistribute digital contents to another in P2P networks. The solution includes three protocols (distribution protocol, usage protocol, redistribution protocol) which are all 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, peer-to-peer, TPM, redistribution, n-time-use digital content**

I. 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. And 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 [1-8]. In this paper, we need such trusted platforms to provide secure environments in which our protocols run. Hence we suppose that an ideal trusted platform has already existed, and how to build such a platform is beyond the scope of this paper (the reader can get more about how we build a trusted platform in our previous work [3, 5]).

And to motivate our work, we make two definitions here: n-time-use digital content and redistribution.

N-time-use digital contents are contents that can be used only n times which is previously defined by the content provider. Moreover, the user can consume them in their own platforms without contacting the provider.

Redistribution is a process in which the content user (sender) sends his or her digital contents to others (receiver). In the P2P architectures, there is no central server which is always online. So we must ensure that the sender and receiver follow the policies of content provider when the redistribution takes place offline. Moreover, if the content is an n-time-use digital content, the problem will become more complex. In this paper, we show how these problems can be solved using minimal trusted hardware functionality provided by TPMs.

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. And finally, we give an informal analysis of our solution's security, and the results of performance experiments.

Outline. The rest of this paper is organized as follows, in Section 2, we provide a summary of those aspects of trusted computing that are relevance to this paper. In Section 3, we identify the security requirements that our solution should satisfy. Then we present our solution in Section 4 and 5. In Section 6, we analyze its security. Following that, the results of performance experiments are given in Section 7. Finally, we conclude with a short summary and future work in Section 8.

Related Work. Securing P2P distribution using trusted computing has already been introduced by [9] for several years. And some concrete schemes of distribution have been proposed based on different models and assumptions [4, 10, 11, 12].

In [10], 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) that supports TPM-based distribution and usage control. 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 contents are not discussed in detail and the security analysis of the solution is not presented.

The authors of [11] present a protocol that allows servers to securely distribute secrets to trusted platforms. They specify the protocol in detail at the level of TPM commands and they informally analyze its security. So the protocol can maintain the confidentiality of secrets in the face of eavesdroppers and careless users. However the distribution model in [11] is based on traditional client-server architectures. So it can not support offline redistributions in P2P architectures.

In [12], the authors identify characteristic P2P scenarios and demonstrate how these can be realized by applying a few basic licensing operations. Then, they present a security architecture to realize these basic license operations. But the distributions of n-time-use digital contents are not discussed, which are common scenarios in P2P architectures.

II. Trusted Computing fundamentals

In this section, we introduce several mechanisms of TPMs which will be used by our solution later. For a comprehensive description, the reader is referred to Trusted Platform Module specifications of TCG [13, 14].

A. Measurement

TCG describes an authenticated boot that the BIOS measures (i.e., cryptographically hashes) the boot loader prior to handing over control, the boot loader measures the operating system loader, and the operating system loader measures the operating system. These measurements' results are stored in platform configuration registers (PCRs) of TPM. Hence the values of PCRs reflect what software stack is in control of the computer at the end of the boot sequence. Moreover, the measurement can also be processed after boot-up in order to get the current environment state [3-5]. An attacker who wants to change the platform configuration without being detected has to corrupt the root of trust for measurement (RTM), which we assume to be infeasible.

B. Protection of Private Keys

The protected storage feature of a TPM allows for the secure storage of TPM keys. The TPM has a storage root key (SRK), which is created upon initialization of the TPM, and is protected in TPM. Then we can create an asymmetric key as children of SRK with TPM, and state its environment for use (PCR values in TCG terminology). The new asymmetric key's private key is encrypted by SRK, and stored outside the TPM. Before we use the private key, it must be loaded into the TPM (decrypted by SRK). Then we can use the private key when the current environment is the one stated while creating the key. Furthermore, each key is either marked as being migratable or non-migratable. In the former case, the key might be replicated and moved to other platforms whereas in the latter case the key is bound to an individual TPM and is never duplicated. If we encrypt digital contents with a migratable key, then the encrypted contents are also migratable. Moreover, we can state the migratable key's PCR values when creating it. Then, the migratable key can be used to decrypt contents only if the state of the platform matches the PCR values.

C. Reporting

Each TPM has an endorsement key (EK) which is a signing key whose public key is certified by a trusted third party (e.g., the TPM manufacturer). However, the EK is only used to obtain a key certificate from a certificate authority (CA) for an attestation identity key (AIK), which is created by the TPM. The AIK is also a signing key whose private key is only used for signing data that has originated in the TPM (for example, signing the PCR values or keys generated by TPM which is also called certifying). We use AIKs instead of EK because of the privacy reasons. A TPM can have many AIKs but only one EK. So using different AIKs to report the PCRs or keys can bring our solution an additional advantage in the privacy protection.

D. Transport Sessions

TPMs can create transport sessions which allows for the grouping of a set of commands into a session. The session provides confidentiality of these commands and can also provide a log of them. In our solution, we will create an exclusive and logged transport session using the AIK as the signing key in order to get an AIK's signing log which will be used as an evidence of some TPM operations. The signing log includes an anti-replay nonce and the inputs, commands, and outputs encountered during the entire transport session. Moreover, by making this transport session exclusive, we ensure that the TPM will not allow other exclusive transport sessions to successfully execute at the same time. This ensures the atomicity of operations in the session [15].

E. TPM Monotonic Counters

A TPM monotonic counter provides an ever-increasing incremental value, which is tamper-resistant, and cannot be reverted back to a previous value. Hence the counters can be used to defend objects (e.g., the data, keys or other objects) in the untrusted storage (e.g., a hard-disk) against

replay attacks. This is very useful for n-time-use digital contents mentioned above. Unfortunately, because the low-cost TPM chip can only afford to have a small amount of internal non-volatile memory, the number of counters is necessarily limited. Specifically, a TPM 1.2 chip is only required to be able to store four independent monotonic counter values at a time, and only one of these counters is usable during a particular boot cycle [15]. So in this paper, we use transport sessions and one TPM monotonic counter to defend n-time-use digital contents against replay attacks by learning from the log-based scheme described in [15-17].

III. Security Requirements

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

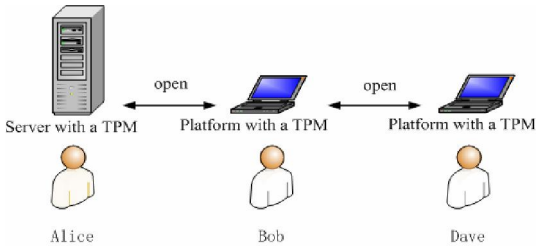

Figure 1. Setting.

Our setting is shown in Figure 1. Alice provides the song s that it is willing to 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 make Bob use s following Alice's rules).

We consider that Bob listens 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 (i.e., Alice believe that dishonest Bob and Dave follow her rules, and do not collude when they listen to or redistribute the song s if their platforms satisfy her security requirements).
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.

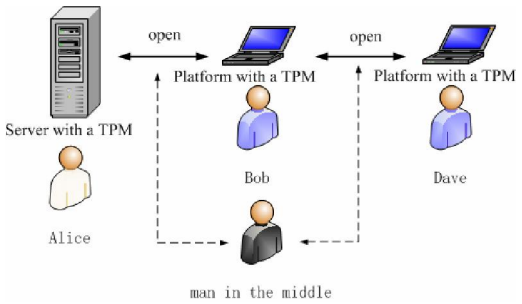

Figure 2. Attacks.

IV. Solution Overview

In this section, we outline our solution which can meet the security requirements 1-3, as listed above, and more details will be described in the following section.

The protected storage feature of a TPM allows for the secure storage of TPM keys, as described in section 2. 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 [11], 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 [14] and protection of private keys, as shown in Figure 3.

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 KB, and certifies public KB using its private AIKB. By verifying the AIK-certified public KB, Platform A ensures that KB 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 KB (for more details, the reader is referred to TCG key migration [14]). 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).

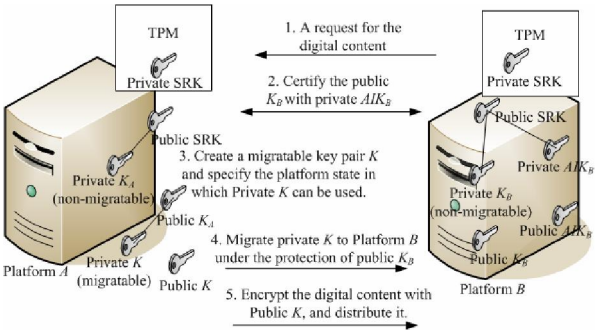

Figure 3. An approach for the digital content (re)distribution with TPM.

If we just consider the security requirements 1 and 2, the approach described above is enough. However, the security requirement 3 need protect the freshness of the n-time-use digital content (i.e., replay attacks should be prevented).

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. In this case, the dishonest user can backup the content before he use it, and replace it with the backup after use in order to get more use times lawlessly. Thus, one solution to this problem would be to employ some form of irreversible state change. 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 the counters in TPMs. So we have to use the transport sessions described in section 2 to bind lots of digital contents to one TPM counter, which is similar to the work in [15-17]. However, the input parameter AntiReplay[14] is replaced with a hash value Hash(public K|flag|nonce) in our approach, so the association between digital contents and TPM counter values can be built directly, which is different from the work in [15-17].

As shown in Figure 4, we suppose that there is already a physical counter in the TPM, which is called CounterA. Then, we establish a transport session, and execute the increment of CounterA, which returns its current value Value(CounterA) 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(CounterA) is bound to the creation, use or migration operation of K. Therefore, the transport session log includes CounterA, Value(CounterA), encryption key of the digital content, flag and a random nonce nonce. We denote this log with TransLogAIK (CounterA, Value(CounterA), public K, flag, nonce). If we find a change of the counter's value without a transport session log, we consider it as an attack. Hence, offline (re)distributions of n-time-use digital contents can be built with these transport session logs, as shown in the following sections.

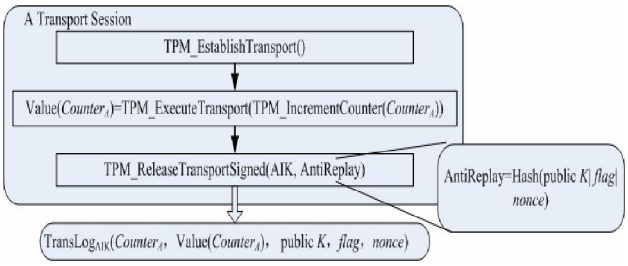

Figure 4. An approach for the freshness protection of n-time-use digital content.

V. Solution Details

In this section we first describe three protocols: distribution protocol, usage protocol and redistribution protocol, which constitute our solution of securing P2P distributions. Then we discuss the anti-collusion approach in which some simple features are added to a TPM.

A. Distribution Protocol

Figure 5 summarizes the distribution protocol in which Alice distributes the n-time-use song *s* to Bob. First, Bob certifies that public K_{Bob} is a non-migratable key with his AIK. Then Alice generates a migratable key pair *K*, which is used to encrypt *s*. And *K* is sealed to a set of PCRs required by Alice, called PCR_{Req}. Therefore, private *K* can only be used to decrypt *s* when the states of Bob's or other peers' platforms match PCR_{Req}. After that Alice migrate *K* to Bob's platform under the protection of K_{Bob}. And then, Bob attests to the current value of $Counter_{Bob}$ by executing TPM_IncrementCounter in a transport session, as described in section 4. Then this transport session log, TransLog($Counter_{Bob}$, *CounterValue*, Public *K*, *created*, *nonce*) is sent to Alice. Alice verifies the log, and then creates a certificate $Cert_{PSK}$ (signed with her key *PSK*), which includes Bob's transport session log. This links the creation of *K* to $Counter_{Bob}$, as well as to a particular point

Value(Counter$_{Bob}$). At last, Alice encrypts *s* with public *K*, and sends it to Bob.

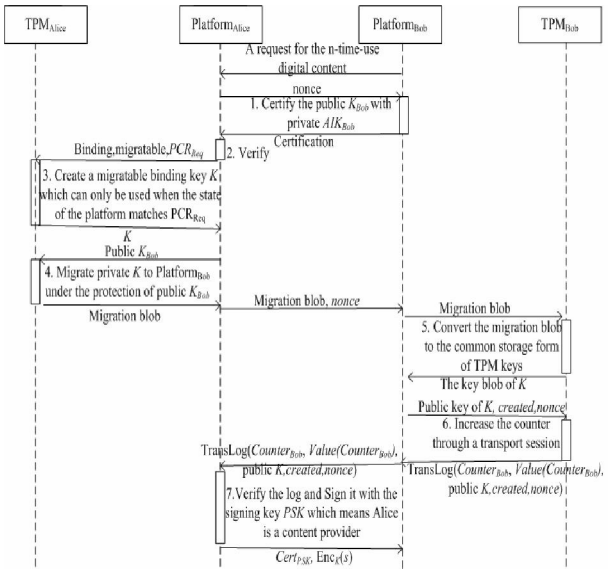

Figure 5. Distribution protocol.

B. Usage Protocol

After Bob get the encrypted digital content Enc$_K$(s), encryption key *K* and usage certificate *Cert$_{PSK}$*, he can use *s* in his platform under the control of some security software *UCS*, which is usually a usage-control system[3-5] represented by a set of PCR values, called *PCR$_{Req}$*. And Alice has already bound *K* to *PCR$_{Req}$* before distributing it. Therefore, private *K* can only be used to decrypt the digital content *s* when *UCS* works well, which means Bob has to be an honest user in a way. *UCS* verifies the transport session logs to find if there is one use time remaining, as shown in Figure 6. And if there are use times left, it calls TPM_IncrementCounter to increase *Counter$_{Bob}$* in a transport session in order to record that one time has been used. At last, Bob listens to *s* for a time.

Note that, the step 2 in Figure 6 can execute successfully only if *UCS* are working well in Bob's system, which will go through the following steps to verify the *Cert$_{PSK}$* and transport session logs:

1. *UCS* verifies the signature of *Cert$_{PSK}$* and distills the *Value(Counter$_{Bob}$)* and public *K* in it.
2. *UCS* uses *AIK$_{Bob}$* to verify all the signatures of transport session logs from *Value(Counter$_{Bob}$)* to the current value of *Counter$_{Bob}$*.
3. *UCS* distills the counter values from the logs, and verifies that all the values are presented from *Value(Counter$_{Bob}$)* to the current value of *Counter$_{Bob}$*.
4. *UCS* extracts a sublist of logs which all contain the public *K* (Note that, there are usually more than one n-time-use digital contents in a platform, and therefore the transport session logs may contain many encryption keys at the same time). So these logs in the sublist are all about the public *K*, and the content *s* encrypted by *K*.
5. *UCS* considers the flag in each log of the sublist, and determines the remaining times of usage of *s*. For example, *UCS* may find that two logs are recorded with "*used*". If the digital content's usage conditions describe it as a 2-time-use content, then *UCS* would reject Bob's request for the content. And if the content is a 3-time-use content, then *UCS* would accept Bob's request, and give the content to Bob.

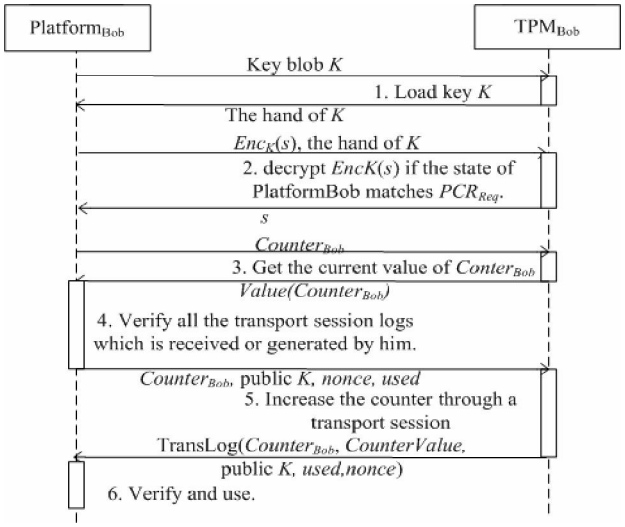

Figure 6. Usage protocol.

C. Redistribution Protocol

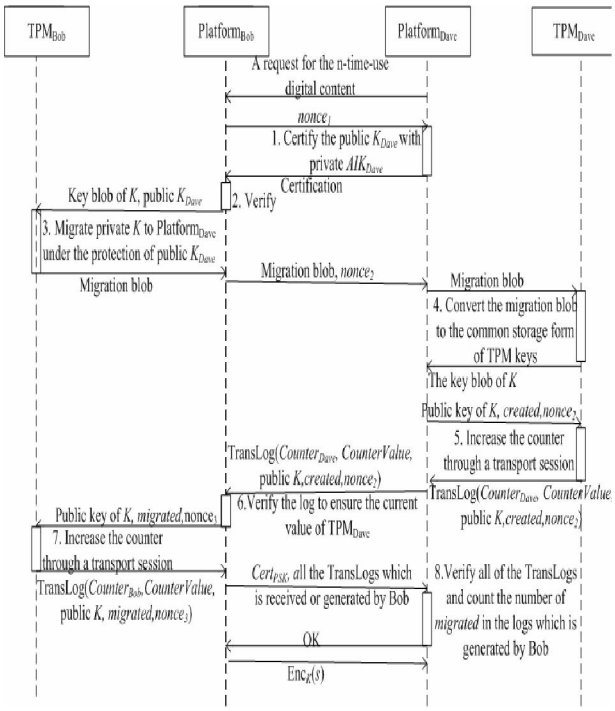

Figure 7. Redistribution protocol.

As depicted in Figure 7, the process of redistribution protocol is similar to the distribution protocol. However, there are three different points between them. The first one is that *K* has been generated and its PCRs have also

been specified by Alice. Bob can not replace it with another key and modify the PCRs because of the protection of TPM. The second one is that Bob has to increase his TPM counter $Counter_{Bob}$ with the flag "*migrated*" through the transport sessions, as shown in step 7 of Figure 7. And the executing times of step 7 represent the number of times which Bob wants to migrate. The last one is that Dave has to verify all the transport session logs which are from Bob. Dave ensures the use times migrated to him by counting the number of "*migrated*" in Bob's transport session logs. These logs which contain "*migrated*" indicate that Bob has spent the use times and these use times are migrated to Dave. And we will give an example of the verification process in

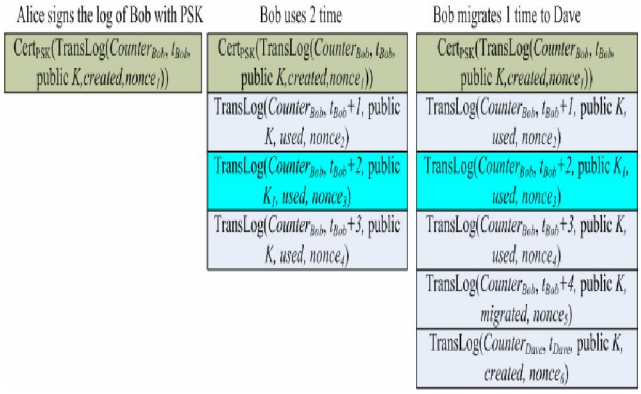

Figure 8. An example of the transport session logs.

Figure 8.

In Figure 8, the vertical column on the left is the $Cert_{PSK}$ signed by Alice, and the vertical column in the middle depicts the transport session logs in $Platform_{Bob}$. Bob uses private *K* to decrypt the digital content for 2 times. This will generate two transport session logs according to our usage protocol (as shown in the middle column, at time t_{Bob}+1 and t_{Bob}+3). It is possible that another encryption key K_1 may be used at time t_{Bob}+2. Then Bob migrates 1 time to Dave, and the transport session logs received by Dave are presented as the vertical column on the right.

These logs can be verified as follows: First, Dave verifies the usage certificate CertPSK and distills tBob. Then he needs to make sure that the current value of CounterBob is tBob +4. After that, he verifies that no violations (loss and forging) have occurred in transport session logs from tBob to tBob +4 which is similar to the verification steps in the usage protocol. Finally, Dave extracts a sublist of logs which contain public K, and counts the number of "used" and "migrated" in them to know how many times the content has been used and how many times the content has been migrated to him. As shown in Figure 8, there is only one log which includes "migrated", and therefore Dave just get one use time of the content from Bob.

D. Anti-collusion Approach

In P2P scenarios, the redistribution takes place offline, so provider Alice can not control the process by contacting users. Hence, two dishonest users may collude. Bob may not do step 7 of the redistribution protocol, and Dave may not verify Bob's logs in order to get more use times.

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 [14]. 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 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 step 7 of redistribution protocol, and verifies the session logs. Note that, because step 7 in Figure 7 does not tie to public *K*, Bob can execute the step with other encryption keys. Then Dave would get the key blob of *K* without any transport session logs about it. However, Dave can not use the usage protocol to decrypt the content in his platform because he can not provide the correct logs of *K*. And if Dave does not execute the usage protocol to verify the logs, the PCR values would not match PCR_{Req}. Hence he can not use private *K* to decrypt the content. Therefore our solution can prevent Bob and Dave from colluding by adding the new feature.

VI. 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. And the replacement of this kind of messages amounts to the Denial of Service attacks, which is beyond our scope of this paper. So our solution meets the security requirement 2, as described in section 3.

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 we assume that the dishonest user lacks tools or expertise to perform more skillful hardware-based attacks, such as snooping on the system's bus, timing analysis. Therefore, the dishonest user can not get the digital content if the state of his platform does not matches the PCR values specified by the provider. And if the platform state is honest, it will execute our usage protocol to get the digital content. Then the replay attacks can be detected by the verification of transport session logs.

And two dishonest peers can collude with each other when redistribution takes place. However, we add two sets of PCRs when generating a key with TPM_CreateWrapKey, as described in section 5. 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 steps of our redistribution protocol, 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 provides security against the dishonest peers' collusion attack. That is, our solution meets the security requirements 1 and 3.

VII. Performance Analysis

We do the performance tests on a PC, with a 2.00 GHZ Pentium 4, and 1 GB main memory. And the TPM chip is from NSM (National Semiconductor Corporation). Figure 9 shows the experimental performance results of key operations in our solution.

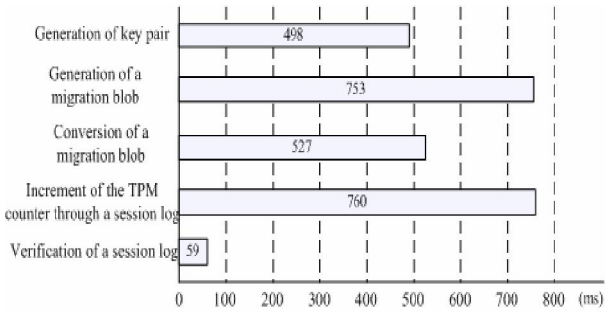

Figure 9. Performance Measurements.

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. However, the cost of verification operations may increase very rapidly with the increment of use times. Fortunately, the use times of an n-time-use digital content are usually less than one thousand. Hence, the cost of verification is acceptable, and our solution is usable in P2P distribution architectures.

VIII. Conclusion and Future Work

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.

In the future, we will improve our prototype in order to integrate it with our usage control system [3, 5]. And its availability and security will be carefully evaluated. Privacy is another important issue. The last column of Figure 8 is the logs which Dave holds. It contains excessive information (the blue-highlighted record). So Dave has a chance to know which encryption keys Bob uses. Although Dave can not exactly know which contents Bob uses, we currently try to prevent Dave from knowing more than he should know.

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] J. Park and R. Sandhu. The UCONabc usage control model. ACM Transactions on Information and Systems Security, 7(1):128–174, February 2004.
[8] A. Pretschner, M. Hilty, and D. Basin. Distributed usage control. Communications of the ACM, (9):39–44, 2006.
[9] 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.
[10] R. Sandhu and X. Zhang, Peer-to-Peer Access Control Architecture Using Trusted Computing Technology. In: SACMAT 2005, Stockholm, Sweden (June 2005)
[11] 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.
[12] 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.
[13] Trusted Computing Group: TCG architecture overview. (TCG Specification)
[14] Trusted Computing Group: TCG TPM specification version 1.2. (TCG Specification)
[15] L.F.G. Sarmenta, M. van Dijk, C.W. O'Donnell, J. Rhodes and S. Devadas. Virtual Monotonic Counters and Count-limited Objects using a TPM without a Trusted OS. 1st ACM Workshop on Scalable Trusted Computing (ACM STC '06). Held at CCS '06, Fairfax, VA, Nov. 2006
[16] L.F.G. Sarmenta, M. van Dijk, J. Rhodes and S. Devadas, Offline Count-Limited Certificates, ACM Symposium on Applied Computing (SAC 2008) Security Track, Fortaleza, Brazil, March 2008.
[17] M. van Dijk, J. Rhodes, L.F.G. Sarmenta, and S. Devadas, Offline Untrusted Storage with Immediate Detection of Forking and Replay Attacks, The 2nd ACM Workshop on Scalable Trusted Computing (ACM STC'07). Held at CCS '07, Alexandria, VA, Nov. 2007.

Hao Li, born in 1983. PhD Information Security(Graduate University of Chinese Academy of Sciences. China. 2011). BA Software Engneering (Xidian University, China. 2005).

He has worked in State Key Laboratory of Information Security, Institute of Software, Chinese Academy of Sciences, Beijing, China. (Research Associate). He is interested in network and system security, trusted computing, trusted storage.

Yu Qin. born in 1979. PhD Information Security(Graduate University of Chinese Academy of Sciences. China. 2009).

He has worked in State Key Laboratory of Information Security, Institute of Software, Chinese Academy of Sciences, Beijing, China. (Research Associate). He is interested in network and system security, trusted computing.

Qianying Zhang. born in 1986. PhD candidate Information Security. She is interested in trusted computing.

Shijun Zhao. born in 1985. PhD candidate Information Security. He is interested in trusted computing.

