TPM-Performance Sensible Key Management Protocols for Service Provisioning in Cloud Computing

Haibo Chen[1], Jun Li[2], and Wenbo Mao[2]

[1] Fudan University
[2] EMC Research China

A Trusted Platform Module (TPM) is a small and hence low-performance hardware chip whose main function — at least for the service provisioning topic of this paper — is to play a trusted third party's role inside a service provisioning computing platform so that the platform will have what we call a behaviour conformity property. The property of behaviour conformity is most needed in service oriented applications, such as utility computing, grid computing and the new notion of cloud computing, where a resource-scarce user (guest) submits jobs to be computed at computational resource providers (hosts). It is inevitable that prior to a session of service provisioning, security protocols will run between the guest, the host, and the TPM. For service provisioning to have scalability albeit TPM's low performance, such a protocol needs to be carefully designed not to place the TPM in a bottleneck position. We propose a protocol mechanism by remodelling the original TPM being the trusted computing base (TCB) into two sub-components: a high performance software TCB which is a measured virtual machine monitor to delegate most of the functions of the TPM, and the original low performance TPM TCB which retains the software measurement function inside itself for low frequent uses. Our result has an independent value for wide deployment of TCG technologies.

1 Problem Statement and Threat Analysis

Fig. 1 illustrates an abstract protocol view of a service-oriented computing problem. Here, Alice is a resource-scarce user who wants to compute her jobs on a resourceful service provider Bob. In the rest of the paper we shall consider Alice as a guest, and Bob as a host. In this service provisioning problem, a guest's process/application running on the host needs protection, both in terms of data/code confidentiality and integrity.

This service provisioning problem has a somewhat unusual threat model. Usually a guest is viewed as a potential adversary which may cause damage to the host and to other guests on the host. However, the unusual direction of the threat we want to emphasize in this paper is the following: the host is a partner of the guest, as well as a potential opponent. There are different ways to view the latter direction of the threat. First, with the host providing services to many guests, it is possible that a guest makes use of the host to launch attacks on other guests. This is particularly possible if the host makes use of commercial-off-the-shelf systems to

B. Christianson et al. (Eds.): Security Protocols 2008, LNCS 6615, pp. 107–114, 2011.

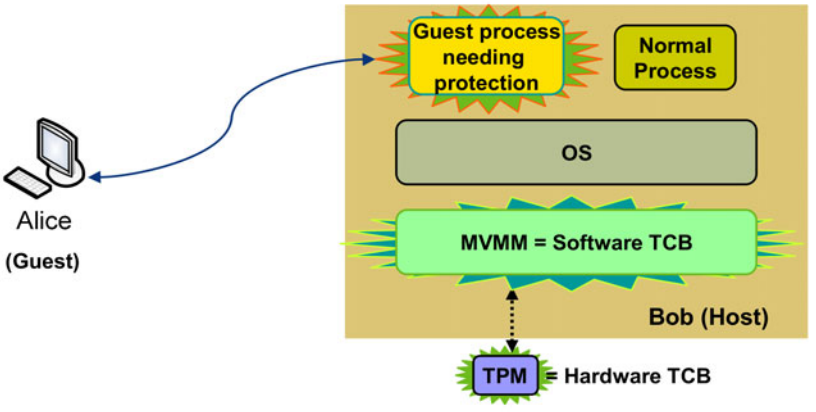

Fig. 1. An abstract protocol structure for service provisioning and host protection (only part of the host components need protection and are marked as "wrapped")

build its service provisioning environment, such as commodity operating systems and third party software which the host has little control about their trustworthiness. Second, the system administrator at the host may also be an adversary, and indeed, with a root-level privilege, the owner adversary is a formidable one! A potential threat from the host to a guest can be unauthorized disclosure of guest's proprietary code/data. This sort of threat is less of a concern in traditional ways of service provisioning which are only affordable from reputable hosts. However, it is nowadays becoming problematic in more and more popular ways of service provisioning like Grid [1] or Cloud [2] computing that service providers are ad-hoc recruited, and the relationship between a guest and a host is in a "one off pay as you go" manner. Serious guest users such as small and medium enterprises and hosts such as seasonally resource abundant financial institutes may not wish to "join the cloud" if strong protection is not in place.

2 Our Contribution: Separation of Trusted Computing Base

We will use Trusted Computing Group Technology to provide the needed protection. In our emphasis on the host provisioning not only computational services, but also strong security services, we use the TPM [3] to provide an integrity measurement on an important and privileged software component: the virtual machine monitor (VMM) [4]. A measured VMM [5] is denoted MVMM which is considered an extension of the TPM as a trusted computing base (TCB). The measurement takes place when the computing platform is booted. The reason we measure a VMM is because this software will be providing most security services, such as process isolation [6,7,8], to materialize the needed property of behaviour conformity at the host.

MVMM is considered the software TCB delegated from the TPM which is the hardware TCB. The separation of software and hardware TCBs is a very

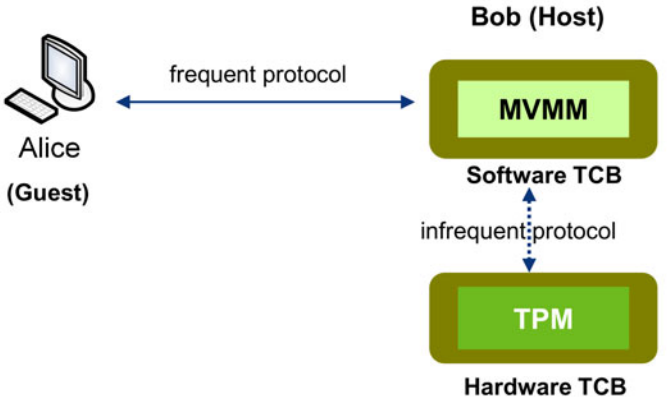

Fig. 2. Protocol separation to minimize the use of the TPM

important and the key point of this paper. The software TCB is in high performance to delegate much of the functions of the hardware TCB which is in low performance. However, as the key point of this paper, there is one important function of the TPM that will not be delegated on to the MVMM: that is the measurement of the very software TCB: one cannot measure itself. Fig. 2 depicts the protocol view abstracted from Fig. 1.

The protocol suite consists of several protocols of which we emphasize two. Most of the time, such as when the host accepts a guest application (most of the time because there can be a large number of guests to be served by one host), a usual protocol (we shall term "frequent protocol") is to run between the guest and the MVMM in the host without involving the TPM. That protocol involves a delegated attestation of the MVMM to the guest. The delegation is possible because MVMM is the software TCB certified by the TPM, and the chain of trust works just in the TCG defined manner.

However, occasionally, a guest application which is already running at the host will enter a state of indecision whether or not the software TCB has been properly measured. These occasions typically include the following situations: to roll-out a guest application to the persistent external storage for processing at a later time (maybe because the host needs to schedule processes on it to achieve a better performance), or to checkpoint a VM image to prevent power loss, among others. We will argue in a moment that in these much rarer occasions, a guest application rolled out to the persistent storage will enter a state of indecision, and we will let the TPM kick in to make the needed judgement. Since the guest application needs "wrapping" (see Fig. 1) outside the memory (inside memory protection is achieved by the memory arbitration function of the MVMM which we will not discuss in this paper), the rolling out of a paused application will involve cryptographic algorithms to be applied to the guest application, such as code/data encryption plus integrity protection. In our design, this rarer case of cryptographic protection is served by applying an "infrequent protocol" which should use key material of the TPM rather than that which has been delegated to the software TCB.

One may ask why the use of the TPM key material is necessary for "wrapping" the rolled out guest applications. Why not just use the delegated key material of the MVMM, which will later permit the MVMM to directly re-admit the rolled out application in an efficient manner? As an important point of notice, we observe the necessary stateless design of the software TCB: the MVMM which has rolled out a guest application and the application to be resumed needs not to be the same session run of the software (may even not be the same version of the software in case software upgrading). Consider the checkpoint case followed by a power loss for instance. The software TCB is rebooted after power back on must already be a new session run. With the stateless design of the software TCB, the new session run (version) of the MVMM already uses a new set of delegated key material, and hence can no longer decrypt the previously rolled out guest application.

3 TPM-Performance Sensible Key Management Protocols

3.1 Notation and Terminology Agreement

To describe the protocol clearly, let's first provide a notational and terminological agreement.

Integrity Measurement: The integrity measurement [9] is a term of TCG technology. In the measurement of a VMM, it is by the TPM computing a cryptographic hash value of VMM code. The measurement can only be done by TPM and will not be delegated to any party external to the TPM. In the TCG specification, an integrity measurement value is stored in a PCR register in the TPM. However in our TPM-performance sensible design, the TPM shall hand a certified copy of the measurement of the software TCB to the software TCB. The certification uses the AIK of the TPM.

Delegation of the software TPM: The delegation takes place in the boot time when the VMM is measured and loaded. The result is that the measured PCR value of the VMM will be digitally signed by the TPM and handed over to the MVMM. Later the MVMM can be attested by a remote guest to verify the delegation using the attestation identity key (AIK) of the TPM. Notice that this step of attestation verification needn't involve the TPM. The MVMM contains a new pair of AIK$'$ with the public part being certified by the AIK of the TPM.

AIK: Attestation Identity Key (AIK) is a TCG term for special purpose public-private key pair inside the TPM for attestation use. In our remodelling, after each measurement of the VMM, a new pair of the session AIK$'$ will be made available to the MVMM which is certified and verifiable by the AIK of the TPM. This AIK$'$ is for the MVMM to run delegated attestation between the MVMM and the guest.

Binding: This is a TCG term for traditional public-key encryption using the public key of a TPM. Decryption using the matching private key inside the TPM is called unbinding. In our intentional design for sensible use of the TPM, this pair of operations will be delegated to the MVMM.

Sealing: This is another TCG term for conditional public-key encryption using a public key of the TPM (usually related to the storage root key SRK). The condition is that encrypted messages are bound to a set of platform metrics PCR values which are specified by the message sender. Decryption using the matching private key inside the TPM requires the TPM to check whether the PCR values inside the TPM and those in the sealing match. Sealing/unsealing operation can only be conducted by the TPM itself and will not be delegated to any party outside the TPM.

3.2 Protocols Description

The protocols description has four parts: (in 3.2.1) the establishment of the software TCB in the boot time (an "infrequent protocol"), (in 3.2.2) the deployment of the guest application which is the frequent case of using delegated functions in the software TPM (a "frequent protocol"), (in 3.2.3) the rolling-out and -in of the guest application which is the infrequent case of using the TPM, and finally (an "infrequent protocol"), and (in 3.2.4) the migration of the guest application (a "frequent protocol"). Of these four cases, only protocols in 3.2.1 and 3.2.3 involve infrequent uses of the TPM (the latter case has only the rolling-in half needs to use the TPM) while the other cases of the cryptographic services should only use the delegated services provided by the software TPM. Throughout the paper in all the figures we shall use dash lines to indicate "infrequent protocol" communications with the TPM, and solid lines, those frequent ones with the software TPM.

In the following protocol descriptions, phase numbers are the line numbers shown in Fig. 3.

3.2.1 The Establishment of the Software TCB

Phase 0: Measurement and delegation

This phase has been described in "integrity measurement", "delegation of the software TPM" and "AIK" paragraphs in 3.1. As we have remarked, this is an "infrequent protocol".

3.2.2 Deployment of the Guest Application

The protection of online guest application involves the first three dash lines in Fig. 3.

Phase 1: Alice attests Bob's platform

In this phase, Alice checks the trustworthiness of Bob's platform by attestation. The attestation is done by MVMM of Bob as follows:

Bob runs a challenge-response protocol with Alice to prove that his AIK′ is certified by the TPM, and he also produces to Alice the signed PCR value of the MVMM. In the run of challenge-response protocol, Alice will verify the AIK′

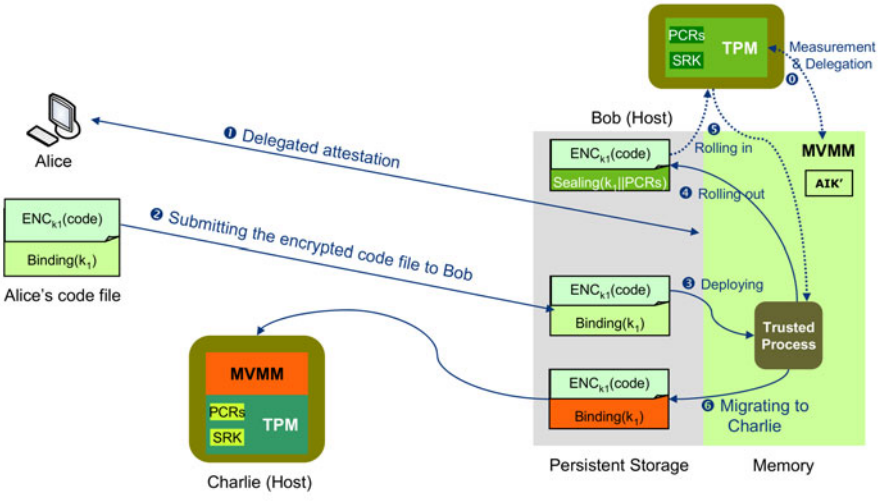

Fig. 3. Key Management Protocols

certificate and check the correctness of the PCR value with respect to the desired VMM code.

If succeeds, Alice should believe that Bob's platform is loaded and running the expected MVMM.

Phase 2: Alice submits encrypted code/data file to Bob

Alice encrypts her code file by using a randomly generated symmetric key k_1, and appends the encrypted file with a meta-data of encrypted k_1 by using the public key of Bob's platform. This encryption is the binding which we described in 3.1.

We know that the encrypted code/data file can only be decrypted by the same MVMM which has been attested and running in the host. We notice that if Bob's platform reboots after receiving encrypted code file, then the new session run of the MVMM will not be able to decrypt the file because each boot will render the new session run of the MVMM to use new key material. This follows the principle of the stateless design of the software TCB.

Phase 3: To load a guest application at the host

When invoking `execve()`, the OS will load the guest program code into the memory. Before transferring the control to the user process, the MVMM will unbind the code.

This protocol is a "frequent protocol".

3.2.3 Rolling Out and in of the Guest Application

As we have discussed, the guest application may need to be rolled out or checkpointed to the persistent storage. We use phase 4 and phase 5 to describe these steps.

Phase 4: To roll out a guest application

If the running guest application needs to be rolled out, the code/data will be saved to the persistent storage. We shall use the sealing function in stead of the binding. To make sense, we formulate the new meta-data as the ciphertext of data "k_1||PCRs" which is encrypted by the public key public key of the TPM.

Phase 5: To roll in a guest application

When the rolled out program code/data need to be rolled back in, the TPM will have to perform the decryption (unsealing). The TPM firstly decrypts the meta-data by using its storage related private key, and then checks if the decrypted PCRs values match the PCR values inside it. If the checking succeeds, the TPM will output k_1 to the MVMM. Otherwise the TPM will output nothing. In this case, we can say that the TPM acts as a protector of empty-headed program code/data.

We remark that only Phase 4 involves "frequent protocol" while Phase 5 is "infrequent".

3.2.4 Application Migration

The application migration is shown in phase 6. If the program is about to be migrated to Charlie, Bob will be in the position of a guest and do the attestation regarding the host Charlie's platform and then prepare for the encrypted code/data file to be deployed to Charlie. This is a regular operation and it uses the same method as the deployment of Alice's code/data to Bob as described in the protocol in 3.2.2.

4 Conclusion and Follow-Up Work

With the very privileged position of the VMM in the bottom of the software stack, it is very reasonable that the TPM delegates most of its performance critical functions on to the measured VMM to make it the software TCB of a high performance. The TPM only retains few critical functions which are directly related to the measurement of the software TCB. Then thoughtful protocol design mechanisms permit the use of the TCG technologies with most performance critical functions delegated to, and served by, the high performance software TCB to minimize the performance penalty related to the typical low performance of the TPM. The particular showcase of the service provisioning use case for Grid and Cloud Computing in this paper actually manifests our work's independent value beyond our particular embodiment: the TCG technology's performance potential needn't be limited by the low performance of the TPM.

A follow-up work for the two protocols designed in this paper can be a formal proof of correctness with respect to a set of carefully deliberated specifications.

References

1. Foster, I., Kesselman, C.: The Grid: Blueprint for a New Computing Infrastructure (1999)
2. Weiss, A.: Computing in the clouds. NetWorker 11(4), 16–25 (2007)

3. Trusted Computing Group. Trusted platform module: TPM Main Specification (2010),
http://www.trustedcomputinggroup.org/resources/tpm_main_specification
4. Goldberg, R.P.: Survey of virtual machine research. IEEE Computer 7(6), 34–45 (1974)
5. Garfinkel, T., Pfaff, B., Chow, J., Rosenblum, M., Boneh, D.: Terra: A virtual machine-based platform for trusted computing. ACM SIGOPS Operating Systems Review 37(5), 206 (2003)
6. Chen, H., Zhang, F., Chen, C., Yang, Z., Chen, R., Zang, B., Yew, P., Mao, W.: Tamper-resistant execution in an untrusted operating system using a virtual machine monitor. In: Parallel Processing Institute Technical Report, Number: FDUPPITR-2007-0801, Fudan University (2007)
7. Chen, H., Chen, J., Mao, W., Yan, F.: Daonity-grid security from two levels of virtualization. Information Security Technical Report 12(3), 123–138 (2007)
8. Chen, X., Garfinkel, T., Lewis, E.C., Subrahmanyam, P., Waldspurger, C.A., Boneh, D., Dwoskin, J., Ports, D.R.K.: Overshadow: a virtualization-based approach to retrofitting protection in commodity operating systems. In: Proceedings of the 13th International Conference on Architectural Support for Programming Languages and Operating Systems, pp. 2–13. ACM, New York (2008)
9. Sailer, R., Zhang, X., Jaeger, T., van Doorn, L.: Design and implementation of a TCG-based integrity measurement architecture. In: Proceedings of the 13th USENIX Security Symposium (2004)