Secure Data Management in Trusted Computing

Ulrich Kühn[1], Klaus Kursawe[2], Stefan Lucks[3],
Ahmad-Reza Sadeghi[4], and Christian Stüble[4]

[1] Deutsche Telekom Laboratories, Technical University Berlin, Germany
ukuehn@acm.org
[2] ESAT – COSIC, KU Leuven, Belgium
klaus.kursawe@esat.kuleuven.ac.be
[3] Theoretische Informatik, University of Mannheim, Germany
lucks@th.informatik.uni-mannheim.de
[4] Horst Görtz Institute, Ruhr-University Bochum, Germany
sadeghi@crypto.rub.de
stueble@acm.org

Abstract. In this paper we identify shortcomings of the TCG specification related to the availability of sealed data during software and hardware life cycles, i.e., software update or/and hardware migration. In our view these problems are major obstacles for large-scale use of trusted computing technologies, e.g., in e-commerce, as adopters are concerned that the use of this technology might render their data inaccessible.

We propose both software and hardware solutions to resolve these problems. Our proposals could be easily integrated into the TCG specification and preserve the interests of involved parties with regard to security and availability as well as privacy.

1 Introduction

The increasing global connectivity and distributed applications both for business and personal use require IT-systems that guarantee confidentiality, authenticity, integrity, privacy, as well as availability. On the technical side, cryptographic and IT security research provide a variety of technical security measures such as encryption and strong authentication mechanisms, firewalls and so forth, to achieve these security targets. However, these measures provide only partial solutions as long as the underlying computing platforms still suffer from security problems.

These issues are addressed by a new generation of computing platforms employing both supplemental hardware and software. Concretely, these initiatives are the TCG (Trusted Computing Group), an IT-industry alliance, and Microsoft's NGSCB (Next-Generation Secure Computing Base); however, so far only the TCG has published specifications [17,16].

According to the TCG the primary goal of this architecture is to improve the security and the trustworthiness of computing platforms [4,5,12,13]. To this end, the conventional PC architecture is extended by new mechanisms to (i)

J.R. Rao and B. Sunar (Eds.): CHES 2005, LNCS 3659, pp. 324–338, 2005.

protect cryptographic keys, (ii) authenticate the configuration of a platform (attestation), and (iii) cryptographically bind confidential data to a certain system configuration (sealing), i.e., the data can only be accessed (unsealed) if the corresponding system can provide the specific configuration for which the data has been sealed.

In this paper we identify shortcomings of the TCG specification related to software and hardware life cycles and propose solutions to resolve these problems. More precisely, we are concerned with the following two problem areas regarding the management of sealed data:

First, the TCG specification defines the sealing functionality in a way that an update or security patch to the trusted computing base can render sealed data inaccessible, even when keeping the same level of security. We propose possible solutions to this problem that preserve security for all involved parties, i.e. owners, users, and remote parties (e.g. content providers), in the sense of multilateral security.

Second, the TCG specification contains a (optional) protocol to partially migrate the TPM-internal data to another TPM of the same vendor (lock-in). However, to our knowledge this protocol is not implemented in any of the existing TPMs. Our proposal allows to securely migrate all data to a different platform without requiring to either tolerate loss of potentially important data or involve all remote parties while at the same time avoiding potential privacy violations.

Our proposals are based on exercising decentralized control of the update or migration procedure where a party trusted by both the system owner as well as remote parties ensures that their respective interests are served.

This paper is structured as follows: In Section 4 we give a summary of the TCG specification as far as needed for the purpose of this paper. We continue in Section 5 with a brief description of the problems regarding the handling of sealed data during software and hardware life-cycles. Section 6 describes a basic system model that we consider suitable to support our solution proposals. In Sections 7 and 8 we describe our proposals for solutions to the software update and the hardware migration problems.

2 Related Work

To our knowledge, only few articles are publicly available that discuss improvements of the TCG specification in the context of platform changes and migration.

In [15] the authors present a security measurement architecture for Linux that measures all executable content upon loading and protects the table of measurements using the TPM's functionality. Remote parties can first verify the integrity of the table of measurements using remote attestation, and can then decide if the current platform configuration is trustworthy. This is applied to remote access where a client configuration is verified before allowing it to access the network [14]. However, platform updates and platform migration is not addressed in [15,14], and, as the PCR values are employed to protect the

current list of measurements, working with sealed data seems difficult, or results in the same problems as discussed in Section 5.

Property-based attestation as proposed in [10,8] extends the remote attestation protocol such that, on a more abstract operating system layer, properties are attested instead of binary representations of platform configurations. While [10,8] focus on remote attestation, we will show in Section 7.3 that this approach is also well-suited for what we call property-based sealing.

3 Conventions

Basic Notation. A (public key) encryption scheme is denoted with the tuple $(\mathsf{Enc}(), \mathsf{Dec}())$ for encryption and decryption algorithms. The tuple (PK_X, SK_X) denotes the public and private key of a party X. Further, a digital signature scheme is denoted by a tuple $(\mathsf{Sign}(), \mathsf{Verify}())$ of signing and verification algorithms. With $\sigma \leftarrow \mathsf{Sign}_{SK_X}(m)$ we denote the signature on a message m using the signing key SK_X. The return value of the verification algorithm $ind \leftarrow \mathsf{Verify}_{PK_X}(\sigma)$ is a Boolean value $ind \in \{true, false\}$. A certificate on a quantity Q with respect to a verification key PK_X is a signature generated by applying the corresponding signing key (not something complex such as a X.509 certificate). A hash function is denoted by $\mathsf{Hash}()$.

Roles. Throughout the paper we refer to the following subjects/roles:

- *Owner*: The owner $\mathcal{O}$ of a system $\mathcal{P}$, e.g. a PC, is an entity who defines, by its security policy $\mathcal{SP}_{\mathcal{O}}$, the allowed configurations of the underlying platform, also including patches/updates. Typical examples are an enterprise represented by an administrator or an end-user owning a personal platform.
- *User*: The user $\mathcal{U}$ of a computing platform $\mathcal{P}$ is an entity interacting with $\mathcal{P}$ under the platform's security policy $\mathcal{SP}_{\mathcal{O}}$. Examples are employees using enterprise-owned hardware; user and owner might also be identical.
- *Remote party*: This refers to any party trying to assess the trustworthiness of a system, e.g. for licensing content to the system's owner. The remote party's security policy $\mathcal{SP}_{\mathcal{R}}$ defines access control rules attached to the content. An example might be a party licensing classified data to the system's owner that wants to ensure that its access policy is also enforced locally.

4 Main Aspects of the TCG Specification

In this section we briefly review the main functionalities of the trusted computing technology proposed in specifications version 1.1b [17] and 1.2 [16] of the *Trusted Computing Group* (TCG).

The main components of the TCG proposal are a hardware component *Trusted Platform Module* (TPM), a kind of (protected) pre-BIOS (Basic I/O System) called the *Core Root of Trust for Measurement* (CRTM), and a support software called *Trusted Software Stack* (TSS) which performs various functions like communicating with the rest of the platform or with other platforms.

Trusted Platform Module. A TPM is the main component of the specification providing a secure random number generator, non-volatile tamper-resistant storage, key generation algorithms, cryptographic functions like RSA encryption/decryption, and the hash function SHA-1[1]. A TPM can be abstractly described by the tuple $(EK, SRK, \mathcal{T})$: the endorsement key EK, an encryption key that uniquely identifies each TPM; the Storage Root Key SRK or Root of Trust for Storage (RTS), uniquely created inside the TPM. Its private part never leaves the TPM, and it is used to encrypt all other keys created by the TPM; the TPM state $\mathcal{T}$ contains further security-critical data shielded by the TPM (see Section 8.1).

The TPM provides a set of registers called *Platform Configuration Registers* (PCR) that can be used to store hash values. The hardware ensures that the value of a PCR register can only be modified as follows: $PCR_{i+1} \leftarrow \text{SHA1}(PCR_i|I)$, with the old register value PCR_i, the new register value PCR_{i+1}, and the input I (e.g. a SHA-1 hash value). This process is called *extending* a PCR.

There are three different types of asymmetric keys a TPM can create:

- *Migratable keys* (MK): Migratable keys are those cryptographic keys that can only be trusted by the party who generates them (e.g. the user of the platform). However, a third party has no guarantee that such key has indeed been generated on a TPM.
- *Non-migratable keys* (NMK): Contrary to a migratable key, a non-migratable key is guaranteed to be kept in a TPM-shielded location. A TPM can create a certificate stating that a key is an NMK.
- *Certified-migratable keys* (CMK): Introduced in version 1.2 of the TCG specification, this type of key allows a more flexible key-handling. Upon creation of such a key with the `TPM_CMK_CreateKey` command, a trusted *Migration Authority* (MA) can be selected, to which decisions to migrate the key are delegated. To migrate a CMK to another platform, the TPM command `TPM_CMK_CreateBlob` expects a certificate of an MA approving of the migration to the destination. Another trusted party, the *Migration Selection Authority* (MSA) may be involved in the process, which only controls the migration, but never gets in contact with any migrated data. In both cases the certificate of the CMK that is used by the owner/user to prove that it was really created by a TPM contains information about the identity of the MA resp. MSA.

Based on this functionality, the TCG specification defines four mechanisms called *integrity measurement*, *attestation*, *sealing*, and *maintenance* which are explained briefly in the following:

[1] SHA-1 [6] generates 160-bit hash values from an input of (almost) arbitrary size. One of the stated security goals for SHA-1 is: finding any collision *must* take 2^{80} units of time. Recently, Wang et. al. [18] claimed an algorithm to find such collisions in time 2^{69}, see also [3]. Though attacking SHA-1 would be challenging, SHA-1 clearly has failed its stated security goals. In contrast to some applications, full collision-resistance is essential in Trusted Computing. Hence, we anticipate revised specifications to switch to another hash function.

Integrity Measurement & Platform Configuration. Integrity measurement is done during the boot process by computing a cryptographic hash of the initial platform state. For this purpose the CRTM computes a hash of ("measures") the code and parameters of the BIOS and extends the first PCR register by this result. A chain of trust is established if an enhanced BIOS and bootloader also measure the code they are transferring control to, e.g. the operating system. The security of the chain relies strongly on explicit security assumptions about the CRTM. Thus, the PCR values $PCR_0, \ldots, PCR_n$ provide evidence of the system's state after boot. We call this state the platform's *configuration*, denoted by $S_i := (PCR_0, \ldots, PCR_n)$.

Attestation. The TCG attestation protocol is used to give assurance about the platform configuration S_i to a remote party. To guarantee integrity and freshness, this value and a fresh nonce provided by the remote party are digitally signed with an asymmetric key called *Attestation Identity Key* (AIK) that is under the sole control of the TPM. A trusted third party called *Privacy Certification Authority* (Privacy-CA) is used to guarantee the pseudonymity of the AIKs. In order to overcome the problem that this party can link transactions to a certain platform, version 1.2 of the TCG specification defines a cryptographic protocol called *Direct Anonymous Attestation* DAA [1], eliminating this CA.

Sealing. Data D can be cryptographically bound to a certain platform configuration S_0 by using the `TPM_Seal` command. Given an asymmetric key pair (SK, PK), we denote this function abstractly with $[D]^{PK}_{S_0} \leftarrow \mathsf{Seal}(S_0, PK, D)$ meaning that D is *sealed for* the configuration S_0. The `TPM_Unseal` command releases the decrypted data only if for the current configuration S'_0 holds $S'_0 = S_0$, or, abstractly, $D = \mathsf{Unseal}([D]^{PK}_{S_i}) \Leftrightarrow ([D]^{PK}_{S_0} \leftarrow \mathsf{Seal}(S_0, PK, D) \wedge (S_i = S_0))$.

According to the current TCG specification [16], `TPM_Seal` only accepts NMKs and it is unclear how the CMKs can be used. As we will see in Section 7 sealing using CMKs would be useful.

Maintenance. The maintenance functions can be used to migrate the SRK to another TPM: The TPM owner can encrypt the SRK under a public key of the TPM vendor using the `TPM_CreateMaintenanceArchive` command. In case of a hardware error the TPM vendor can extract the encrypted SRK from the maintenance archive, decrypt it, and load it into another TPM.

Unfortunately, the maintenance function is only optional and, to our knowledge, not implemented by currently available TPMs. Furthermore, the maintenance function works only for TPMs of the same vendor.

5 Problem Description

The integrity measurement mechanism securely stores the platform's initial configuration into the registers (PCRs) of the TPM. Any change to the measured software components results in changed PCR values, making sealed data inaccessible under the changed platform configuration. While this is desired in the case

of an untrustworthy software suite or malicious changes to the system's software, it may become a major obstacle for applying patches or software updates. Such updates do generally not change the mandatory security policy enforced by an operating system (in fact, patches *should* close an existing security weakness not included in the system specification). Nevertheless, the altered PCR values of the operating system make the sealed information unavailable under the new configuration.

We see here a major issue with the current TCG proposal, since the semantic of the sealing operation is too restrictive to efficiently support sealed information through the software life-cycle including updates / patches. The main problem is the lack of a mapping between the security properties provided by a platform configuration and its measurements. This difficulty is also pointed out in [13]: "[...] to recognize which reported PCR values were good, given the myriad platforms, operating system versions, and frequent software patches".

A further problem we see with the TCG's proposal is how to handle hardware replacements in a computing platform. Such replacements are necessary due to outdated or faulty hardware. In corporate contexts, hardware is typically replaced every few years. Any sealed data bound to a given TPM cannot directly be transferred to another TPM, because it is encrypted with a key protected by the SRK, which in turn is stored within the TPM. While this is intended to prohibit unauthorized copying, it also effectively prohibits the owner of a system to migrate the data to a replacement hardware.

In our opinion the existing mechanisms offered by the TCG specification are insufficient:

Shortcomings of Certified Migratable Keys: CMKs allow platform owners to migrate keys to another platform (see Section 4). Unfortunately, migration authorities (MA) have to explicitly certify every single key that the platform owner may want to migrate. Thus, complete migration of a TPM to another platform would require an enormous amount of certificates and thus traffic. In our opinion, it is unreasonable to assume that migration authorities can guarantee both availability and security of their services under these conditions. Another drawback is that CMKs cannot be used as encryption keys for the sealing operation. Thus, data bound to a configuration cannot be migrated using this approach.

Shortcomings of Maintenance Procedure: Although the maintenance function defined in the TCG specification protects platform owners against loss of data, its current instantiation is unsatisfying: First, this mechanism is only optional and not implemented until now. Second, the maintenance function requires interaction with the TPM vendor. In our opinion, this fact is problematic from both availability and free-enterprise perspectives: First, in case of a hardware failure all data controlled by the TPM is affected, thus TPM vendors are in the position to ask for high fees. Moreover, the TPM owner is lost if the TPM vendor does not exist anymore. Second, the maintenance function does not allow platform owners to migrate to a TPM of a different vendor.

6 Basic System Model

In this section, we propose an abstract system model that provides a practical solution to the missing link problem between the security properties offered by a platform and its configuration.

Typically, remote parties offering content do not want to limit the usability of the content to only one platform configuration. Instead, they require that their policy $\mathcal{SP}_{\mathcal{R}}$ attached to the content cannot be circumvented. Since such policies can become rather complex, we propose a three-layered security architecture to enforce them:

- **Application layer:** Remote parties can attach a piece of restricted code, a *policy checker*, to their content that decides whether the requirements of the policy $\mathcal{SP}_{\mathcal{R}}$ are fulfilled or not. Hence it is not necessary to define a general policy language be used by all remote parties. Instead, remote parties can explicitly or implicitly code the policy to be enforced into the policy checker. Note that the purpose of the policy checker is similar to what is called a *trusted viewer* [7]: The policy checker verifies that the underlying platform fulfills the necessary properties, and can enforce a complex security policy.
- **Operating-system layer:** The operating system layer performs all tasks of a usual operating system that cannot violate security policies of the involved parties. This includes resource sharing (e.g. filesystems and user interfaces) and non-critical device drivers.
- **Security-kernel layer:** As the policy checker enforces the security policies, the underlying TCB "only" has to guarantee that unauthorized entities cannot manipulate the platform such that enforcement mechanisms can be bypassed. This includes a strict separation between applications (isolation). Here the sealing mechanism provided by the underlying trusted computing hardware helps to ensure these elementary security properties. The security kernel has to be trusted by all involved parties.

Obviously existing monolithic operating systems are not capable of fulfilling these security requirements (e.g. they do not at all provide a secure isolation between processes). We therefore suggest a small Trusted Computing Base (TCB) that offers the properties of the security-kernel layer. Examples of such architectures are [2,9,11].

The advantage of this architecture is that the PCR values used by the underlying trusted computing technology (e.g. sealing, attestation) depends only on the code of the security-kernel. Applications, e.g. a web server, and the operating system layer can use more abstract services like property-based attestation [10,8] provided by the TCB. This way changes of these higher layers (e.g. due to patches) do not change the PCR values, keeping the architecture more flexible.

The reader should note that the software-based proposals in the following sections assume the existence of such a security architecture providing a trusted computing base that *securely* isolates different processes from each other. This implies that the solutions do not translate to legacy operating systems, e.g. Linux or Microsoft Windows, without weaker security guarantees.

7 Platform Updates

In our system model, updates of components outside of the TCB are easy as no PCR values are affected. However, as discussed in Section 5, a configuration change of the TCB involves more work. We consider the following security and usability requirements that must be fulfilled by a computing platform providing the sealing functionality:

- **Security.** A platform of configuration S_0 can access sealed data $[D]_{S_0}^{PK}$ only if the attached security policy $\mathcal{SP}_{\mathcal{R}}$ defines S_0 to be trustworthy. This represents the interests of the remote parties.
- **Availability.** Information sealed to a platform enforcing the security policy $\mathcal{SP}$ should be available under all platforms that are capable of enforcing $\mathcal{SP}$. Thus, a software patch should not make the information inaccessible.

In the following, we propose three solutions to the platform update problem. The first one is based on an extended software function offered by the operating system layer and allows to reuse existing TPM's. The second solution (Section 7.2) extends the TCG specification by a TPM command but allows more flexible handling of sealed data. In Section 7.3 we discuss the advantages of a property-based sealing mechanism and show how it can be implemented.

7.1 Software-Supported Updates

To guarantee availability of sealed data when the TCB's configuration S_i is changed to a S_j offering the same security properties, the TCB must provide a service we call *Update Manager* (UM). The main task of the UM is to seal the data for the new configuration S_j *before* the TCB update happens. Note that the UM has to be invoked whenever components of the TCB are to be changed.

There are several requirements on the UM to correctly update sealed data:

1. The sealed data $[D]_{S_i}^{PK}$ must be available to the update manager, i.e. in a central store, so that the UM can re-seal them. Alternatively the TCB could be implemented such that it uses only one sealed cryptographic key to encrypt all data under configuration S_i. As a consequence, only one sealed key has to be updated by the UM.
2. The PCR values for the new configuration S_j must be known to the UM. This implies that the binary representation of the module to be updated is available such that the UM can pre-calculate the expected configuration.
3. Only data that is sealed for S_i can be updated. This implies that the UM cannot update data sealed for a different configuration S_i'.
4. The update manager must have some means to ensure that the new configuration S_j offers the same security properties as the old one with respect to $\mathcal{SP}_{\mathcal{R}}$. Several solutions to this problem are imaginable. We suggest to introduce a trusted party that is responsible for certifying that two configurations S_i and S_j offer the same security properties. The identity of the trusted party could be, e.g., hard-coded into the TCB resulting in a unique

platform configuration that depends on this party. Thus, by sealing data for that configuration, remote parties explicitly agree with the TTP selected by the platform owner.
5. The whole process must be failsafe, i.e. the process must recover if it is disturbed, e.g. due to user-interruption or power-failure.

Remark 1. The underlying assumption of our model is that the TCB is small and independent enough such that updates appear only rarely. However, on a system not conforming to our system model, such as typical monolithic systems, severe problems may occur: First, system updates occur frequently, since the TCB is very complex and device drivers are part of it. Second, the UM would need to ensure that the update process does not violate any policy required by the respective remote party. Therefore, the UM has to know which security assurances are necessary for which sealed data. Currently, there is no means in the TCG proposal to specify the required assurances for sealed information.

The update process proposed above has the drawback that data can only be updated if it can be accessed under the current configuration. This complicates updates of core components like the BIOS or the bootloader, since sealed data for every possible configuration needs separate handling *before* the new component is installed. The solutions in the next two sections avoid this shortcoming.

7.2 Hardware-Supported Updates

Our second proposal is based on a new TPM command `TPM_UpdateSeal` that re-seals data for another platform configuration based on an *update certificate*. This TPM command works independently of the current platform configuration. Thus, it is possible to update all sealed data under one configuration, regardless of whether the current configuration can access the sealed data. We expect that such a command could be easily integrated into existing TPM designs.

We assume that a trusted party called *Update Certification Authority* (UCA) with a key pair (SK_{UCA}, PK_{UCA}). It issues update certificates $cert_{update} := \mathsf{Sign}_{SK_{UCA}}(S_i, S_j)$ that vouch for configurations S_i and S_j offering the same security properties.[2] Obviously, the UCA has to be trusted by all involved parties to fulfill the requirements of a secure update function. In the sense of multilateral security, both the user/owner and the remote party have to agree on an UCA *before* data is sealed. Our solution can be applied in two different ways:

– The TPM internally binds the identity $I_{UCA} = \mathsf{Hash}(PK_{UCA})$ of the UCA to the CMK key pair (SK, PK). By means of a certificate (a signature) on (PK, I_{UCA}) by the TPM, remote parties can verify that the data encrypted under this key can be updated based on certificates of that UCA. As this is based on certified migratable keys (CMKs), using a UCA instead of an MA, we require them to be usable as sealing keys.[3]

[2] A UCA could be, e.g., an existing authority already involved in software certification.
[3] Here a new type of updatable sealing key could be introduced that has exactly this functionality.

– Remote parties can define the UCA that is allowed to update data they have sealed by securely binding an identifier of the UCA to the data to be sealed (like the platform configuration under which the data can be accessed).

To provide multilateral security, the user or platform owner should be participating in defining the UCA. Otherwise, remote parties could force them to accept an untrusted one. Therefore, we prefer the first approach.

We now specify the proposed TPM command. Let the following entities and quantities be given: a TPM, a UCA with public key PK_{UCA}, and an update certificate $cert_{update} := \mathsf{Sign}_{SK_{UCA}}(S_i, S_j)$.

As prerequisites, let the user/owner have identified a UCA by $I_{UCA^*} = \mathsf{Hash}(PK_{UCA^*})$ upon creation of a new CMK key pair (SK, PK) used for sealing, and let the CMK certificate $cert_{cmk}$ state that PK is generated by a valid TPM and that it can be updated based on certificates issued by UCA^*.

Command specification (`TPM_UpdateSeal`).

Parameters: $[D]^{PK}_{S_0}$ sealed with PK bound to I_{UCA^*}, $cert_{update}$, PK_{UCA}

Command Description: The TPM

1. checks the update certificate's validity: $\mathsf{Verify}_{PK_{UCA}}(cert_{update}) \stackrel{?}{\rightarrow} \mathsf{true}$.
2. checks that $\mathsf{Hash}(PK_{UCA}) = I_{UCA^*}$.
3. checks $cert_{update}$ covers $[D]^{PK}_{S_0}$, i.e. that $S_0 = S_i$.
4. returns an error if any of the above checks fails,
5. computes and returns $[D]^{PK}_{S_j}$.

This proposed TPM command allows users/owners to control who is responsible for the creation of update certificates. Further, it is multilaterally secure with regard to remote parties who can decide if they are willing to accept the UCA.

7.3 Property-Based Sealing

While the TCG-specified trusted boot process allows to efficiently detect changes to the code, it neither allows any conclusion if a certain set of PCR values corresponds to a trustworthy system, nor does it provide any evidence if a change in the values represents a property-preserving update or an attempt to subvert the system. Since the software-supported and the hardware-supported update function are based on this so-called *binary attestation*, they both have the drawback that sealed data has to be re-sealed whenever the platform's configuration intentionally changes.

In [10,8], *property-based attestation* builds on attestation of abstract properties instead of binary representations of the platform configuration. Informally, a property of a platform describes an aspect of its behavior with respect to certain requirements, such as security-related requirements, e.g., that a platform has built-in measures for Multi-Level Security (MLS) mechanism, or built-in privacy preserving measures conforming to privacy laws; or, more suitable to our context here, a property could be the fact that the TCB guarantees the secure

execution of a policy checker (see Section 6). In general a property can be viewed as a model, and any platform complying to this model is said to provide this property.

Providing *property-based sealing* allows to bind data to properties instead of hash values of binaries. This approach has several advantages: First, if data is only bound to an abstract property, it is no longer necessary to re-seal the data during software upgrade if the underlying property does not change. Second, the UCA (see Section 7.2) would only certify that a software release provides a certain property instead of issuing update certificates between each two such releases. Third, property-based sealing allows users to use sealed data under several different platform configurations providing the same properties. Fourth, remote parties do not have to care about the concrete platform configuration of the user, since they only have to bind the data to the desired property. Fifth, remote parties are unable to discriminate certain platform configurations (e.g. Open-Source software) since the concrete configuration is kept secret.

In practice, property-based services can be provided by a small TCB (see Section 6) that depends on conventional binary attestation. As a result the large amount of applications that are using this service (e.g. a web server) do not have the problems with sealed data discussed in Section 5. In Appendix A we describe a possible realization of property-based sealing based on update certificates.

The difficulty with property-based attestation and sealing is to define the concrete semantics of a property, different remote parties may desire to bind their data to different properties, and a concrete platform configuration may provide properties that do not exactly match those desired properties. Our system model (see Section 6) moves the handling of complex property analyses to policy checker executed on the application-layer, so the only remaining property the TCB has to provide is to guarantee a secure execution of the policy checker.

8 Migrating to Another Hardware Platform

In Section 5 we discussed the shortcomings of the currently-specified maintenance mechanism. To remedy this we propose to extend the TCG specification by a multilateral-secure migration mechanism that fulfills the following requirements:

- **Completeness:** Platform owners should be able to *move* the secret state $\mathcal{T}_s$ (see Section 8.1) of a source TPM_s to a destination TPM_d. This implies that the source TPM is reliably cleared afterwards, so that only a single instance of the state exists.
- **Security:** Migration from TPM_s to TPM_d should only be possible if TPM_d has the same level of security as TPM_s.
- **Fairness:** The specification must not dictate the involved parties which TTP defines the security relations between different TPMs. Moreover, migration should be possible without the need to interact with the TPM vendors.

To fulfill these requirements, we suggest reasonable modifications and extensions of the current TCG specification, among them that all security-critical

TPM-data can be securely extracted in a non-discriminating manner. Section 8.2 describes how the idea underlying certified migratable keys (CMKs) can be used in a fail-safe protocol to securely migrate the state of a TPM to another one that provides the same security properties.

8.1 Sharing the TPM's State

Migration of a TPM's state to another TPM is only meaningful if all security-critical parts of the TPM's state can be migrated. The maintenance mechanism allows platform owners to export TPM-protected storage, including the SRK and the owner's authorization data. Unfortunately, the current TCG specification [16] does neither define mechanisms to securely export the state of the non-volatile (NV) memory, nor the values of the monotonic counters (MC). Currently, if migration is an issue, these protected resources cannot be used to store security-critical data.

Therefore we suggest to extend the TCG specification such that all resources offered by the TPM can be exported in order to make migration possible.[4]

8.2 Migration Protocol

The purpose of our migration protocol is to allow platform owners to migrate a TPM's contents while not breaking the security guarantees it provides. Our migration protocol is based on the idea of making Storage Root Keys (SRK) migratable under tightly controlled circumstances similar to Certified Migratable Keys (CMK): For this, we introduce a trusted party called *TPM Migration Authority* (TMA). Its purpose is to decide about the migration of the SRK; the TMA shall be bound to the SRK upon its creation by the `TPM_TakeOwnership` command.

The decision whether a TPM provides at least the same level of security as another TPM is highly security-critical since if TPM owners were capable of migrating their data to a less secure TPM, remote parties could not trust TPMs at all. Therefore, a TMA needs to be trusted by both the platform owner and remote parties. In practice, a TMA could be an institution that does security evaluation and certification. For privacy reasons the choice of a TMA should remain with the involved parties only. Since remote parties need to know this TMA, we further suggest to include the TMA's identity (e.g. a hash of its public key) into the AIK certificates.

The parties involved are a source $TPM_s = (EK_s, \mathcal{T}_s, SRK_s)$, a destination $TPM_d = (EK_d, \mathcal{T}_d, SRK_d)$ and a TMA identified by PK_{TMA}. We explicitly assume that the complete TPM state $\mathcal{T}$ including NV and MC can be extracted encrypted under the TPM's SRK (see Section 8.1).

[4] This would also solve another problem stemming from the NV and MC being limited resources: An operating system might allocate all these resources, so that other operating systems installed on the same system would be precluded from using them.

We assume as prerequisites:

- Upon taking ownership of TPM_s, a TMA is identified by $\mathsf{Hash}(PK_{TMA^*})$.
- The owner of TPM_s has obtained from the TMA a migration certificate $cert_{mig} = \mathsf{Sign}_{PK_{TMA}}(\mathsf{Hash}(EK_{TPM_s}), \mathsf{Hash}(EK_{TPM_d}))$ on the hashes of the endorsement keys of the TPMs. To do so, the owner proves the authenticity of both TPMs by sending the corresponding vendor certificates on TPM to the TMA.

Then the migration protocol consists of the following steps, involving a new command $\mathsf{Migrate}()$:

1. The owner extracts the encrypted TPM state $C_{\mathcal{T}_s} := \mathsf{Enc}_{PK_{SRK_s}}(\mathcal{T}_s)$ (see Sect. 8.1).
2. The owner extracts the endorsement key EK'_{TPM_d} from the destination TPM_d.
3. Upon invocation of $\mathsf{Migrate}(cert_{mig}, EK_{TPM_d}, PK_{TMA})$, TPM_s checks that
   - $cert_{mig}$ is valid, i.e. $\mathsf{Verify}_{PK_{TMA}}(cert_{mig}) \rightarrow \mathrm{OK}$,
   - the TMA issuing $cert_{mig}$ has the correct identity, i.e. $\mathsf{Hash}(PK_{TMA}) = \mathsf{Hash}(PK_{TMA^*})$,
   - the contents of $cert_{mig}$ is consistent with its endorsement key EK_{TPM_s} resp. the supplied EK_{TPM_d} using the respective hash values.
4. TPM_s encrypts SRK_s under EK_d, yielding $C_{SRK_s} := \mathsf{Enc}_{EK_d}(SRK_s)$. This is sent to the platform with TPM_d.
5. TPM_s switches into a persistent state that allows only two TPM commands: The command `TPM_ExtractMigrationData`, which returns C_{SRK_s} (the SRK encrypted under TPM_d's endorsement key), and the command `TPM_OwnerClear`, which deletes the state $\mathcal{T}_s$ and SRK_s.
6. The encrypted TPM state and SRK, i.e. $C_{\mathcal{T}_s}$ and C_{SRK_s}, are loaded into TPM_d. If an error occurs, steps 5 and 6 can be repeated.
7. After successful migration to TPM_d the owner invokes `TPM_OwnerClear` on TPM_s to clear its state and SRK.

9 Summary and Conclusion

In this paper we addressed problems arising from management of sealed data with respect to software as well as hardware life-cycles that result from the current TCG specification. In our view these problems are major obstacles for large-scale use of trusted computing technologies, e.g. in e-commerce applications.

Our proposed solutions to both problem areas are based on the principle of multilateral security and decentralized control, where only the involved parties agree on trusted parties that help to mediate between the interests of the involved parties, without a central authority like the TCG prescribing a trusted party. Furthermore, this principle protects the privacy of a system's owner and user as well as the security interests of remote parties.

For the problem of maintaining availability of sealed data after a software update we proposed a several solutions, one purely in software, a second by augmenting the TPM with an additional command, and a third one based on abstract (security) properties. While the software solution would work with current hardware, it imposes several strict requirements on the operating system design. The other solutions would open up room for advanced trusted computing concepts using abstract security properties.

Our proposal for a hardware migration method would allow to move the contents of one TPM to another one providing multilateral security. It requires some changes to the TPM, but these should be easy to integrate into the design while respecting the security interests of all involved parties.

To conclude, our proposals can resolve the identified shortcomings in the current TCG specifications regarding management of sealed data during software and hardware lifecycles. We suggest to introduce them into the TCG standardization process.

An interesting line of research might be to design protocols that employ zero-knowledge techniques so that a platform owner and remote party can agree on update and migration authorities without revealing the actual authority. Here a remote party could issue a list of authorities it trusts, and the platform owner gives a zero-knowledge proof of membership for the authority of his choice.

References

1. E. Brickell, J. Camenisch, and L. Chen. Direct anonymous attestation. In *Proceedings of the 10th ACM Conference on Computer and Communications Security*, Washington, DC, USA, Oct 2004. ACM Press.
2. P. England, B. Lampson, J. Manferdelli, M. Peinado, and B. Willman. A trusted open platform. *IEEE Computer*, 36(7):55–63, 2003.
3. A. K. Lenstra. Further progress in hashing cryptanalysis, February 2005. `http://cm.bell-labs.com/who/akl/hash.pdf`.
4. Microsoft Corporation. Building a secure platform for trustworthy computing. White paper, Microsoft Corporation, Dec. 2002.
5. C. Mundie, P. de Vries, P. Haynes, and M. Corwine. Microsoft whitepaper on trustworthy computing. Technical report, Microsoft Corporation, Oct. 2002.
6. National Institute of Standards and Technology (NIST), Computer Systems Laboratory. Secure hash standard. Federal Information Processing Standards Publication (FIPS PUB) 180-1, Apr. 1995.
7. National Research Council. *The Digital Dilemma, Intellectual Property in the Information Age*. National Academy Press, 2000.
8. J. Poritz, M. Schunter, E. Van Herreweghen, and M. Waidner. Property attestation—scalable and privacy-friendly security assessment of peer computers. Technical Report RZ 3548, IBM Research, May 2004.
9. A.-R. Sadeghi and C. Stüble. Taming "trusted computing" by operating system design. In *Information Security Applications*, volume 2908 of *Lecture Notes in Computer Science*, pages 286–302. Springer-Verlag, Berlin Germany, 2003.
10. A.-R. Sadeghi and C. Stüble. Property-based attestation for computing platforms: Caring about properties, not mechanisms. In *The 2004 New Security Paradigms Workshop*, Virginia Beach, VA, USA, Sept. 2004. ACM SIGSAC, ACM Press.

11. A.-R. Sadeghi, C. Stüble, and N. Pohlmann. European multilateral secure computing base - open trusted computing for you and me. *Datenschutz und Datensicherheit DuD, Verlag Friedrich Vieweg & Sohn, Wiesbaden*, 28(9):548–554, 2004.
12. D. Safford. Clarifying misinformation on TCPA. White paper, IBM Research, Oct. 2002.
13. D. Safford. The need for TCPA. White paper, IBM Research, Oct. 2002.
14. R. Sailer, T. Jaeger, X. Zhang, and L. van Doorn. Attestation-based policy enforcement for remote access. In *Proceedings of the 10th ACM Conference on Computer and Communications Security*, Washington, DC, USA, Oct. 2004. ACM Press.
15. R. Sailer, X. Zhang, T. Jaeger, and L. van Doorn. Design and implementation of a TCG-based integrity measurement architecture. In *Proceedings of the 11th USENIX Security Symposium*. USENIX, Aug. 2004.
16. Trusted Computing Group. TPM main specification, Version 1.2, Nov. 2003. `http://www.trustedcomputinggroup.org`.
17. Trusted Computing Platform Alliance (TCPA). Main specification, Feb. 2002. Version 1.1b.
18. X. Wang, Y. L. Yin, and H. Yu. Collision search attacks on SHA1, February 2005. `http://cryptome.org/sha-attacks.htm`.

A Property-Based Sealing Using Update Certificates

In practice, properties could be represented by a random but fixed value, while the mapping $value \rightarrow property$ defines the property assigned to that value. These values can be used in the sealing process to define the PCR values S^* of *virtual configurations* which now describe properties instead of concrete binary systems. If the `TPM_UpdateSeal` command (or an extension to `TPM_Unseal` with similar functionality) is available, it can be employed to translate between a property P_i and a concrete configuration S_i. This translation would work as follows:

- Remote parties seal data for a property P_i represented by the virtual configuration S^*, along with information which UCAs are allowed to certify that a concrete configuration actually implements the security properties, resulting in a sealed blob $[D]^{PK}_{S^*}$.
- Given a configuration S_i that actually implements the security properties of S^*, one obtains a certificate stating this fact. This certificate has the same format as the update certificates of Section 7.2, i.e. $U = \mathsf{Sign}_{SK_{UCA}}(S^*, S_0)$.
- The `TPM_updateSeal` command is used to translate $[D]^{PK}_{S^*}$ into a sealed blob $[D]^{PK}_{S_0}$ which can then directly be used.

This way, also the update problem for patched system software would just vanish, as all that is necessary to update to another concrete configuration S_j also implementing the properties of S^* is to obtain a certification of this fact. Only $[D]^{PK}_{S^*}$ would be kept in long-term storage, possibly caching $[D]^{PK}_{S_0}$.