Security evaluation of scenarios based on the TCG's TPM Specification ⋆

Sigrid Gürgens[1], Carsten Rudolph[1], Dirk Scheuermann[1], Marion Atts[2], and Rainer Plaga[2]

[1] Fraunhofer – Institute for Secure Information Technology SIT
Rheinstrasse 75, 64295 Darmstadt, Germany
{guergens,rudolphc,scheuermann}@sit.fraunhofer.de
[2] Federal Office for Information Security (BSI), Godesberger Allee 185-189, 53175 Bonn, Germany
{rainer.plaga,marion.atts}@bsi.bund.de

Abstract. The Trusted Platform Module TPM is a basic but nevertheless very complex security component that can provide the foundations and the root of security for a variety of applications. In contrast to the TPM, other basic security mechanisms like cryptographic algorithms or security protocols have frequently been subject to thorough security analysis and formal verification. This paper presents a first methodic security analysis of a large part of the TPM specification. A formal automata model based on asynchronous product automata APA and a finite state verification tool SHVT are used to emulate a TPM within an executable model. On this basis four different generic scenarios were analysed with respect to security and practicability: secure boot, secure storage, remote attestation and data migration. A variety of security problems and inconsistencies was found. Subsequently, the TPM specification was adapted to overcome the problems identified. In this paper, the analysis of the remote attestation scenario and some of the problems found are explained in more detail.

1 Introduction

The Trusted Platform Module TPM as specified by the Trusted Computing Group TCG [3] provides basic security mechanisms like protected storage areas, computation of cryptographic functions and attestation of integrity measurement. The TPM specification is highly complex and it is diffcult to verify that secure architectures can be based on it. Consequently, (and rather unsurprisingly), errors, inaccurate descriptions and inconsistencies were found in previous versions of the specification. However, as the TPM is supposed to be a trust and security anchor it is necessary that the specification defines a secure TPM. Previous work on analysing the security of the TPM has so far only covered small

⋆ This work was initialized and funded by the German BSI (Federal Office for Information Security)

details. The goal of the work presented in this paper was to provide a first systematic, scenario-based security analysis covering large parts of the specification. A variety of problems was found and documented. All of them were discussed within the TCG (in particular with the TPM and Infrastructure working groups) and resulted in changes, amendments and corrections to revision 94 and newer versions of the TPM specifications.

In the first step of the project, TPM-based protocols were developed for four generic scenarios, namely secure boot, secure storage, remote attestation and data migration. The particular sequences of TPM commands were documented in detail and simulated using the SH Verification Tool [8, 5]. Subsequently, attack scenarios were developed for all scenarios and the previously developed protocols were analysed with respect to these attack scenarios. Finally, the study was completed by analysing several generic attack scenarios and general, scenario-independent issues concerning TPM commands and data structures. The chosen analysis approach was a combination of formal, tool-supported analysis and thorough review by security experts. The scenarios use the different TPM authorisation protocols (e.g. object independent OIAP and object specific OSAP) and allowed to consider the interplay of the different TPM commands and security mechanisms for the security analysis.

In the following sections we give a short overview of the TPM and the automata model for the analysis. Because it is impossible for space reasons to elaborate on our results of all four generic scenarios we chose a concrete "remote attestion" scenario as an illustrative example to demonstrate the analysis approach and to explain some of the problems that were found.

Finally, the mitigation strategies and changes to the TPM specification that resulted from the problems are explained.

2 A short introduction to TPM technology

A TPM [3] usually is implemented as a chip integrated into the hardware of a platform (such as a PC, a laptop, a PDA, a mobile phone). A TPM owns shielded locations (i.e. no other instance but the TPM itself can access the storage inside the TPM) and protected functionality (the functions computed inside the TPM can not be tampered with). The TPM can be accessed directly via TPM commands or via higher layer application interfaces (the Trusted Software Stack, TSS).

The TPM offers two main basic mechanisms: it can be used to provide evidence of the current state of the platform it is integrated in and applications that are running on the platform, and it can protect data on the platform (such as cryptographic keys). For realizing these mechanisms the TPM contains a crypto co-processor, a hash and an HMAC algorithm, a key generator, etc.

In order to prove a certain platform configuration, all parts that are engaged in the boot process of the platform (BIOS, master boot record, etc.) are measured (e.g. integrity check hash values for software) by a so-called root of trust for measurement (RTM) on the platform, and the final result of the accumu-

lated hash values is stored inside the TPM in a so-called Platform Configuration Register (PCR). An entity that wants to verify that the platform is in a certain configuration requires the TPM to sign the content of the PCR using a so-called Attestation Identity Key (AIK), a key particularly generated for this purpose. The verifier checks the signature and compares the PCR values to some reference values. Equality of the values proves that the platform is in the desired state. Finally, in order to verify the trustworthiness of an AIK's signature, the AIK has to be accompanied by a certificate issued by a trusted Certification Authority, a so-called Privacy CA (P-CA). Note that an AIK does **not** prove the identity of the TPM owner.

Keys generated and used by the TPM have different properties: Some (so-called non-migratable keys) can not be used outside the TPM that generated them, some (like AIKs) can only be used for specific functions. Particularly interesting is that keys can be tied to PCR values (by specifying PCR number and value in the key's public data). This has the effect that such a key will only be used by the TPM if the platform configuration (or some application) is in a certain state (i.e. if the PCR the key is tied to contains a specific value). In order to prove the properties of a particular key, for example to prove that a certain key is tied to specific PCR values, the TPM can be used to generate a certificate for this key by signing the key properties using an AIK.

TPM keys have a defined structure whose one part contains public information like the key's public part or binding to specific PCR values. The sensitive part of the key contains information like the key's private part and authorization data. It is encrypted by the key's parent key, whose sensitive information is again encrypted by its parent, thus constituting a key hierarchy whose root is the so-called Storage Root Key (SRK).

In order for any key to be used by a TPM (e.g. for decryption), the key's usage authorization value has to be presented to the TPM. This together with the fact that the TPM specification requires a TPM to prevent dictionary attacks provides the property that only entities knowing the key's authorization value can use the key.

Non-migratable keys are especially useful for preventing unauthorized access to some data stored on the platform. Binding such a key to specific PCR values and using it to encrypt data to be protected achieves two objectives: The data can not be decrypted on any other platform (because the key is non-migratable), and the data can only be decrypted when the specified PCR contains the specified value (i.e. when the platform is in a specific secure configuration and is not manipulated).

3 Description of a concrete "remote attestation" scenario

The following concrete scenario is used throughout the rest of paper for explanation of the security analysis that was originally carried out on more generic scenarios. The example scenario reflects a typical corporate IT environment where the use of TPMs can actually improve security processes. We consider two em-

ployees Bob and Alice. Bob's PC is equipped with a Trusted Platform Module TPM protecting information on the PC and measuring the PC's configuration. Alice's PC may or may not be equipped with a TPM. The TPMs of the company are owned by the company's IT administration (ITadmin), i.e. the process of "taking ownership" of the TPM was executed by the IT administration and the TPM's Owner Authorisation value is only known to ITadmin. Bob and ITadmin are having local (physical) and remote access to Bob's PC.

The main goal of using the TPM in this is to give Alice the ability to bind data to a particular configuration of Bob's PC. This way Alice shall be assured that her security requirements are satisfied by Bob's PC although it is not under Alice's control. These requirements concern data Alice wants to make available to Bob, in particular Alice has the following requirements:

- The data shall only be available on a platform determined by herself (i.e. on Bob's PC). Therefore, initially during the attestation a particular identity key (actually providing pseudonymity) associated with a particular platform (i.e. with TPM) is fixed and the data shall not be available on any other platform.
- Furthermore, Alice requires that the data shall only be readable by Bob, i.e. that no other actor (neither on Bob's PC nor on any other platform) shall be able to read the data without Bob's or Alice's consent.

This results in the security requirement that the data shall be confidential for Alice and Bob, i.e. nobody else shall ever get to know the data, in particular it shall not be known by ITadmin.

We use the following assumptions:

1. Alice trusts the Privacy CA used for attestation and trusts a TPM certified by P-CA in that it acts according to the specification.
2. Processes on a specifically configured platform PF (i.e. Bob's PC) and Alice's own PC do not make available any data to other processes, platforms, platform users or devices.
3. Alice trusts Bob not to deliberately or accidently make the data available to others.
4. Bob does not make available any key usage authorization data he knows to other TPM users or to ITadmin.

To prohibit that the data leaves Bob's PC it is configured in a specific way known to Alice. The main mechanism to provide platform configuration information is by using the TPM_Quote command and sending the result, i.e. current PCR information signed with a certified identity key AIK. This process is one of the basic TPM mechanisms. More details on TPM_Quote can be found in part 3 of the TPM specification [2]. One practical example of secure boot with description of the attestation can be found in the work of Sailer et al. [11]. Additionally, in order to enforce the security policies, Alice needs to ensure that the configuration is still the same when data is decrypted by the TPM. This can be achieved by binding a key to particular PCR values that allow the TPM

to use it only after having checked that platform and system state meet certain conditions, and to use this key for encryption of the data.

This can be achieved as follows:

- Alice requires Bob to provide a TPM generated key (more precisely, the public part of such a key) that is non-migratable and bound to certain PCR values reflecting the correct state of Bob's PC. Additionally Alice requires a certificate for this key in order to be able to verify the key's properties.
- Alice uses this key to encrypt the data she wants to make available to Bob.
- For decrypting the data, Bob loads the key into his TPM, the TPM returns a key handle and then Bob can reference the key through the key handle and use the TPM for decryption. Only if his PC is in the state acceptable to Alice the TPM will actually decrypt the data.

The message sequence (MSC) chart in Figure 1 shows a more detailed description of the command sequences.

4 Security evaluation using Asynchronous Product Automata (APA)

In this section we introduce the Fraunhofer SIT approach for security evaluation. We model the entities of a system using asynchronous product automata (APA). APA are a universal and very flexible operational description concept for cooperating systems [9]. It "naturally" emerges from formal language theory [8]. APA are supported by the SH-verification tool (SHVT) that provides components for the complete cycle from formal specification to exhaustive analysis and verification [9].

Asynchronous product automata (APA) and the SHVT have been successfully used in the past to model and analyse the security of cryptographic protocols (see for example [6] or [5]). This section gives an introduction in the previous work on cryptographic protocols that provides the foundations for the security validation of TPM based scenarios.

4.1 Specification of cryptographic protocols using APA

An APA can be seen as a family of elementary automata. The set of all possible states of the whole APA is structured as a product set; each state is divided into state components. In the following the set of all possible states is called state set. The state sets of elementary automata consist of components of the state set of the APA. Different elementary automata are "glued" by shared components of their state sets. Elementary automata can "communicate" by changing the content of shared state components.

Figure 2 shows a graphical representation of an APA for a system that contains Bob's PC and TPM and the TPM owner ITadmin. The boxes are elementary automata and the ellipses represent their state components.

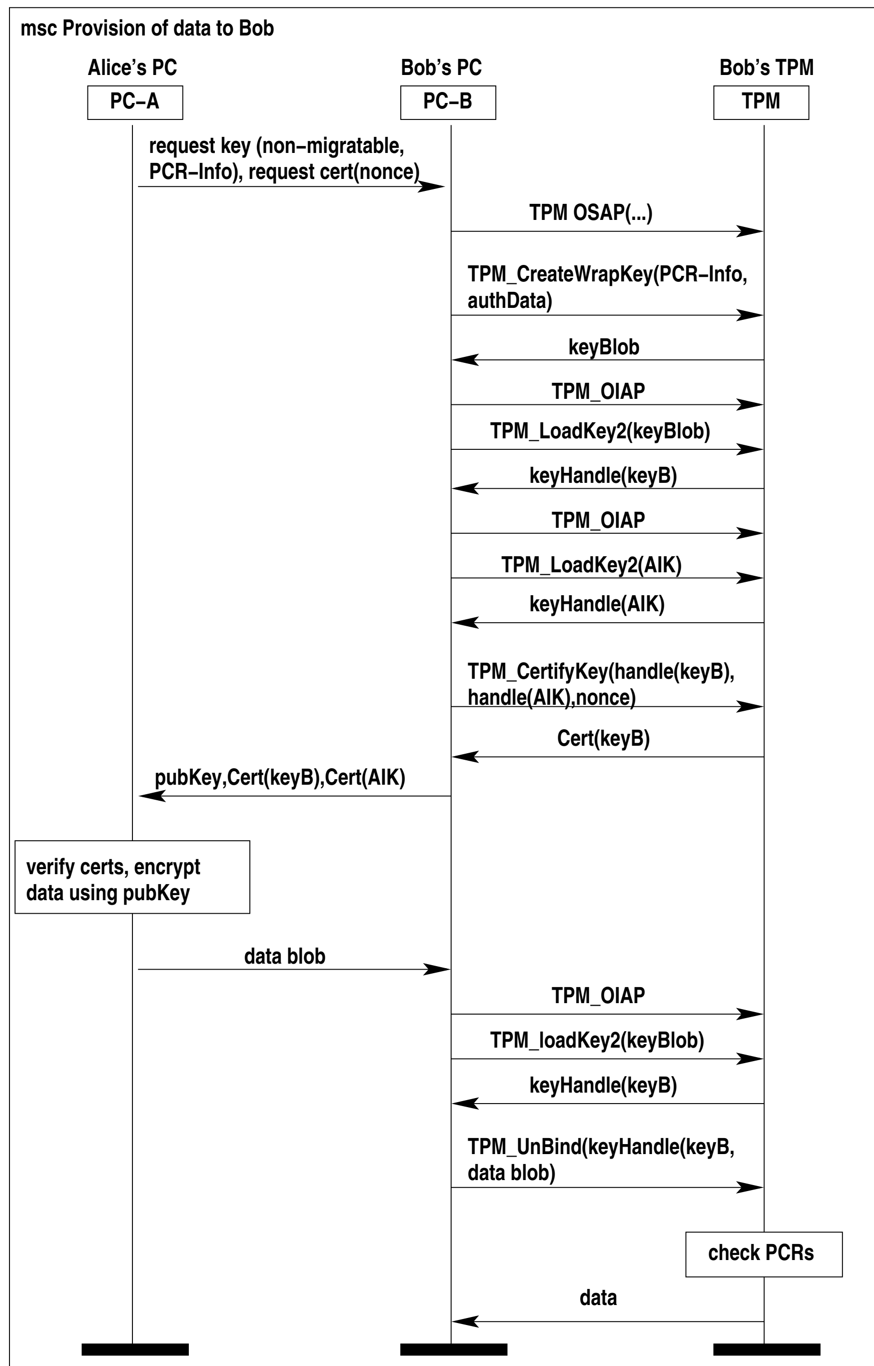

Fig. 1. MSC visualising a sequence of TPM commands

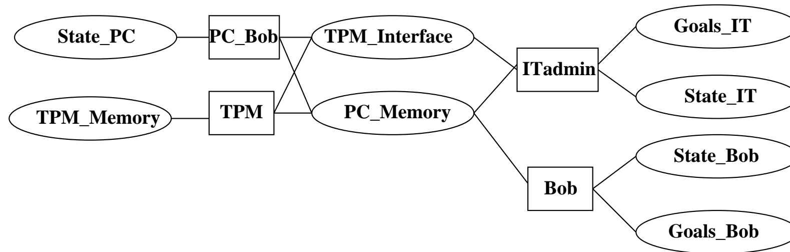

Fig. 2. APA model with Bob's PC and TPM and ITadmin

The neighbourhood relation N (graphically represented by an arc) indicates which state components are included in the state of an elementary automaton and may be changed by a state transition of this automaton. For example, automaton TPM may change TPM_Interface but cannot read or change the state of State_IT.

Each protocol participant P is modelled by an appropriate number of elementary automata that perform the agent's actions, accompanied by an adequate number of state components. What is appropriate depends on the environment and the attack model that shall be modelled. In order to reduce the state space it is often convenient to use one elementary automaton per agent. In the model shown in Figure 2, each agent (i.e. the TPM, Bob, Bob's PC and ITadmin) owns a state component for storing keys and other data specifying its current state. The state component Goals is used to add predicates describing goals the agents shall reach with the protocol (see Section 4.2).

The figure shows the structure of the APA. Communication of the automata is achieved by adding to and removing data from shared state components. By providing PC_Bob and ITadmin with access to TPM_Interface and PC_Memory (modelling storage space on Bob's PC) we model the fact that both agents have access to the TPM and to Bob's PC. Also, TPM has access to these shared components.

The full specification of the APA includes the state sets (the data types), the transition relations of the elementary automata and the initial state, which we will explain in the following paragraphs.

State sets, messages and cryptography The basic message sets, symbolic functions, messages, state component sets etc. depend on the scenario to be analysed. For what is needed to analyse classical protocols like Needham-Schroeder (symmetric and public key) [7], Otway-Rees [10] etc., we refer the reader to [5]. For protocols using other cryptographic primitives and needing other attack models, the definitions can be adapted easily. Examples of more complex protocols that have been analysed include fair non-repudiation protocols [6]. In the following we will concentrate on the details concerning the analysis of TPM scenarios.

State sets of components For the definition of the domains of the state components as well as for the definition of the set of messages, we use as basic sets a set of natural numbers, a set of agents' names, a set of random numbers and nonces, a set of constants. We use key names and flags to model various different key types, and a set of security predicates on global states. For TPM analysis, we additionally use a set containing all TPM structures (see [1] for their specification).

The union of all sets of agents, constants, etc., represents the set of *atomic messages*, based on which we construct a set $\mathcal{M}$ of messages by concatenation and application of symbolic functions.

Symbolic functions We use a system of formal cryptographic primitives. This allows to specify a system in a very realistic way because all variables of the protocol are considered. A good system of axioms is necessary to represent a cryptographic model as real as possible. Each of these symbolic functions models one specific kind of cryptographic system. An HMAC for example (HMACs are used to protect TPM commands and answers) is defined by $HMAC(key1, data1) = HMAC(key2, data2)$ implying $key1 = key2$ and $data1 = data2$. All these properties together define for each $m \in \mathcal{M}$ a unique shortest normal form (up to commutativity). The set $\mathcal{M}essages$ is the set of all these normal forms of elements $m \in \mathcal{M}$.

Now elements of $\mathcal{M}essages$ constitute the content of the state components except the agents' Goals component which contains predicates on global states. For the formal definition of state sets and further symbolic functions we refer the reader to [5].

State transition relations and initial state To specify the agents' actions we use so-called *state transition patterns* describing state transitions of the corresponding elementary automaton. As an example of a state transition pattern, Table 1 shows the specification of a state transition where the TPM receives the command TPM_OSAP. We use internal state components *new_nonce* and *new_handle* to model the generation of nonces and session handles by the TPM.

The lines above $\stackrel{\mathbf{TPM}}{\rightarrow}$ indicate the necessary conditions for automaton TPM to transform a state transition, the lines behind specify the changes of the state. $\hookrightarrow$ and $\hookleftarrow$ denote that some data is added to and removed from a state component, respectively. TPM does not perform any other changes within this state transition. The syntax and semantics of state transition patterns for APA is explained in more detail in [4].

Finally, the initial state has to be specified. It contains in particular all data stored in TPM_Memory when taking ownership. We do not go into further detail here.

The above specification can now be executed in the SH Verification Tool. Thus, the correctness of the specification can be verified and the behaviour visualised. The following section explains how this specification is extended to include malicious behaviour in order to evaluate the security of the TPM command sequence.

(TPM_OSAP)	Pattern name
$(X, entityType, entityValue, entity, nonceOddOSAP,$ $nonces, nonceEven, nonceEvenOSAP, sharedSecret,$ $authSecret, authHandle)$	Variables used in the pattern
$(entityValue, entity) \in TPM_Memory,$	TPM checks that entity the OSAP shall authorize is loaded
$\xrightarrow{\text{TPM}}$	state transition is performed by TPM
$(X, TPM, ('TPM_OSAP', entityType, entityValue,$ $nonceOddOSAP)) \hookleftarrow TPM_Interface,$	OSAP command is removed from TPM_Interface
$authSecret := get_authData(entityType, entity),$	entity auth value is assigned to authSecret
$nonces \hookleftarrow new_nonce,$ $authHandle \hookleftarrow new_handle,$	Nonce (handle) list is removed from $new_nonce(new_handle)$ and assigned to $nonces$ $(authHandle)$
$nonceEven := head(nonces),$ $nonceEvenOSAP := head(tail(nonces)),$	$nonceEven/nonceEvenOSAP$ are assigned a new nonce
$tail(tail(nonces)) \hookrightarrow new_nonce,$ $tail(authHandle) \hookrightarrow new_handle,$	Rest of the nonces (handles) are returned to new_nonce (new_handle)
$sharedSecret := hmac(authSecret,$ $(nonceEvenOSAP, nonceOddOSAP)),$	$sharedSecret$ is assigned the OSAP secret
$(TPM_OSAP, head(authHandle), nonceEven,$ $nonceEvenOSAP, sharedSecret) \hookrightarrow TPM_Memory,$	TPM stores session secret and nonces inTPM_Memory
$(TPM, X, (head(authHandle), nonceEven,$ $nonceEvenOSAP)) \hookrightarrow TPM_Interface$	TPM returns authHandle and nonces to caller

Table 1. TPM receives command TPM_OSAP

4.2 Security goals and malicious behaviour

Security goals In our model, the state components Goals are used to specify security goals. Whenever an agent P performs a state transition after which a specific security goal shall hold from the agent's view, a predicate representing the goal is added to the respective Goals component. Note that the content of Goals has no influence on the system behaviour, i.e. on the occurrence of state transitions.

A system is secure (within the scope of our model) if a predicate is true whenever it is element of a Goals component. In the SH Verfication Tool, a generic break condition continuously evaluates all predicates in Goals during the computation of the reachability graph. The computation is stopped as soon as an attack is found, i.e. one of the predicates is not satisfied in the current state.

For the analysis of the scenario introduced in Section 3, it is, for example, necessary to consider confidentiality. The **confidentiality** of a message m in state s can directly be decided by looking at the content of state components. As long as the dishonest agent (or agents) does not own the cleartext of m, this message can be considered confidential. In order to express that agent P considers message m to be confidential, the predicate $(P, conf, m)$ is added to $Goals$.

Introducing dishonest behaviour In order to perform a security evaluation, our model includes the explicit specification of dishonest behaviour, i.e. behaviour that contradicts the protocol specification. For each type of dishonest behaviour, the APA includes one elementary automaton with the respective state components and state transition relations for specifying the concrete actions. The behaviour of the attack APA depends on the attack model we want to use. As will be explained in Section 5, various different scenarios have to be considered, e.g. whether or not the attacker has access to the TPM of the agent to be attacked. Our APA model can capture any combination of these scenarios.

The elementary automaton of a dishonest agent is specified according to the attack model to be considered. This usually includes the ability of the automaton to read all tuples from state components it has access to, to extract any parts from the tuples, add them to its knowledge and use this to construct new tuples and to add these to state components it has access to. It also includes the ability to apply symbolic functions to messages it knows and to decrypt ciphertext it knows provided it knows the necessary key as well. For the particular TPM attack model the elementary automaton of a malicious agent knows all TPM command structures and can generate TPM commands and other messages by concatenation of data it knows.

5 TPM evaluation

5.1 General considerations

In this section we discuss the various attack possibilities that have to be considered for scenario based on TPM functionality. They are concerned with the question of who knows key authorization data as well as with questions related to particular TPM commands.

The specific intentions of an attacker may be different: First of all, an attacker may be interested in getting access to certain secret data, i.e. in breaking data confidentiality, e.g. authorization data necessary to execute certain TPM commands. Another attacker's strategy may consist of breaking data integrity. In this case, he does not directly want to get access to certain secret data but just wants to forge certain data protected by the TPM or the contents of a TPM command causing the TPM to perform another action than the one intended by the honest actor. Furthermore, an attacker may just want to be able to use the TPM for certain actions he is not authorized to perform, independently of any specific confidential data.

Before using a key, the TPM requires authorization data for this key. The derivation and quality of this authorization data is not specified by the standard. Therefore, one has to assume that an attacker can get hold of one or more authorization data. Thus independently of a specific scenario the consequences of combinations of the following cases have to be considered:

1. An attacker E knows TPM owner authorization.
2. An attacker E knows SRK authorization.
3. An attacker E knows the authorization data of the key to be used.
4. An attacker E knows the authorization data of the key to be used plus authorization data for all keys in the above key hierarchy.
5. An attacker E does not know any key authorization data.
6. An attacker E owns/does not own another TPM2.
7. An attacker E has access to the TPM.
8. An attacker E has no access to the TPM, but to PF.

Knowing authorization data An attacker knowing the TPM owner authorization may execute certain TPM commands explicitly reserved for the owner. This includes the possibility of changing the owner authorization data and the authorization data for the SRK. By doing this, the use of all TPM keys is blocked for other users and the TPM owner. This is particularly relevant if an application is using the TPM and key usage authorization fails. Knowing the SRK authorization alone enables an attacker to load a large amount of keys since the SRK is typically used as a parent key for many other keys. However, it is not sufficient to change the SRK authorization data.

If an attacker only knows the authorization data of a key other than the SRK he can use the key with the TPM (e.g. for decryption of data) provided that the key is already loaded to the TPM. For being able to also load a key, he additionally needs to know the authorization data of all keys in the above key hierarchy up to a key already loaded (or the SRK). Knowing a key's and its parent's authorization data and being able to load the parent enables an attacker to change the key's authorization data, again with the effect that the key can not be used by anybody else. Attacks based on the knowledge of key authorization data are highly relevant for breaking data confidentiality.

Attacks without knowledge of authorization data An attacker not knowing any authorization data will not be able to directly access any confidential data or to directly execute any specific TPM functions. He may just be able to forge data or manipulate communication between an honest actor and the TPM where no authorization data are needed (like changing PCR values). In case the attacker knows the public part of a TPM storage key and has access to the platform, he can generate data or key blobs using this key and substitute other data or key blobs that have this key as parent. Even if the attacker does not know any public TPM key but has access to the platform, he can substitute blobs by appropriate data he knows from earlier activities. This is highly relevant for breaking the data integrity of key or data blobs.

Attacks with additional TPM Owning another TPM2 enables an attacker to perform other specific malicious actions: By placing his TPM2 in a state as desired by the honest actor he can give the actor the impression that the target TPM is in the desired state although it is not. In general, this attack strategy may be followed independent of having access to TPM, and it is especially interesting in combination with manipulation of the platform. Such attacks are especially relevant for the scenarios of Secure Boot and Attestation where the control of the platform state plays an important role.

5.2 Evaluation of the scenario

For the analysis of the example scenario we assume that Bob and Alice have determined a unique AIK to use for attestation. In our attack model we consider the case where ITadmin acts dishonestly and tries to get hold of the data intended for Bob. We concentrate on the situation where ITadmin does not use a further TPM. Other attack models are possible, the attacker may be some outside entity having or not having access to Bob's PC and TPM and owning or not owning another TPM. The corresponding APA model is shown in Figure 3.

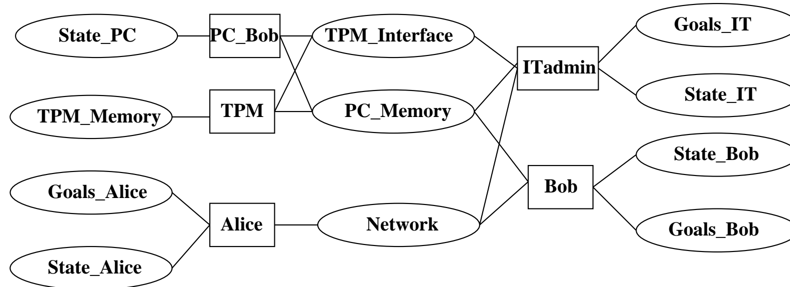

Fig. 3. APA attack model: ITadmin attacks Bob

Manipulation of TPM key handles The aim of ITadmin is to be able to decrypt the data intended for Bob. Recall that Bob needs to transfer the public part of his key *keyB* to Alice which is used by Alice to encrypt the data. According to the MSC shown in Section 3, Bob first creates *keyB*, then loads it, and then lets the TPM generate a certificate for this key, using the AIK both Bob and Alice agreed on. We assume that ITadmin does not own this AIK, i.e. does not know its authorization data. This allows the following to happen:

- Bob (i.e. Bob's PC) loads *keyB* and receives the key handle.
- Bob loads his AIK and receives its key handle.
- Bob sends TPM_CertifyKey command including the following parameters:

  - key handles of AIK and of $keyB$,
  - Alice's nonce,
  - HMAC based on the usage authorization data of the AIK, according to the TPM specification Part 3 [2] not covering the key handles,
  - HMAC based on the usage authorization data of his own key $keyB$, also not covering the key handles.

– ITadmin blocks this command.
– ITadmin then loads its own key $keyIT$, receiving a key handle for this key.
– Then ITadmin removes the HMAC authorizing $keyB$ and the key handle for $keyB$ from the TPM_CertifyKey command and adds the key handle for its own key $keyIT$ and a new HMAC authorizing this key.
– Now the TPM generates a certificate not for $keyB$ but for $keyIT$, using Bob's AIK, and returns this certificate, accompanied by an HMAC based on the AIK authorization data and a second HMAC based on the authorization data of $keyIT$.
– ITadmin blocks this message again, thus Bob's PC will never receive any reply (it would not accept the certificate anyway since it expects the second HMAC to be based on the authorization data of Bob's own key $keyB$ which ITadmin can not generate).
– ITadmin can now send the public part of $keyIT$ to Alice, accompanied by a certificate generated with an AIK not owned by ITadmin.
– Alice accepts this key since it has the required properties and since Alice has no way of knowing who actually owns the key.

Why does the TPM accept the manipulated command TPM_CertifyKey? In general HMACs that protect the integrity of the commands never cover the key handles that are used in the command. The reason for this lies outside of the actual TPM specification. It is motivated by what is called virtualization of key handles: Because of memory restrictions, the TPM can not handle more than a certain amount of keys at a time. Thus the TPM might have to delete keys that are still needed by an application. Each time a key is loaded again it usually is assigned a different key handle. So in order to release higher level applications from the task of having to re-load keys and to handle different key handles, these applications just use one virtual key identifier. The process of re-loading keys and managing the relation between this one identifier and the changing key handles is kept by lower layer software, for example by the TPM Software Stack (TSS), that provides an API to the application on one hand and communicates with the TPM on the other hand. Nevertheless, applications shall be able to construct and check HMACs for TPM commands. As this cannot be done with the virtual key handle, key handles are not included.

Since the key handles of TPM_CertifyKey are not covered by the HMACs, exchanging the key handle for $keyB$ and the corresponding HMAC does not invalidate the HMAC corresponding to the AIK. Hence both HMACs are valid and the TPM processes the command. In Section 6 we will explain how this attack can be avoided.

Integrity of key blobs As explained in Section 2, keys generated by the TPM have a certain structure. The public part contains information like the key being or not being migratable, like PCR values the key may be bound to, it contains the actual public key part, etc. The sensitive part is encrypted using the key's parent and contains the private key part, the key authorization data, etc. However, this structure does not allow to identify the key other than by its public key part. Hence the integrity of a key can not be protected by TPM functionality, i.e. if no additional measures are taken the exchange of one key by another key that has the same parent will not be noticed.

We assume now that between the creation of Bob's key and its loading into the TPM some time passes, i.e. that meanwhile the key is stored in PC_Memory. We assume further that ITadmin knows the parent key of *keyB* (e.g. because it is the SRK). Thus ITadmin can generate its own key blob for some key *keyIT* using the parent key of *keyB*. It can now substitute the key blob of *keyB* stored in PC_Memory by its own, unnoticed by Bob (and Bob's PC). Now the following command sequence can happen:

– Bob (i.e. Bob's PC) loads ITadmin's destination key *keyIT* instead of his own and receives the key handle.
– Bob loads his AIK.
– Bob sends TPM_CertifyKey command including the key handles of his AIK and of *keyIT*, Alice's nonce, an HMAC based on the usage authorization data of the AIK, again not covering the key handles, and an HMAC based on the usage authorization data of his own key *keyB*, also not covering the key handles.
– ITadmin blocks this command, removes the HMAC authorizing *keyB* and adds a new HMAC authorizing its own key *keyIT*.
– The rest of the process is as described above: TPM generates a certificate for *keyIT*, using Bob's AIK, and returns this certificate.
– ITadmin blocks this message again and then can send *keyIT* to Alice, accompanied by a certificate generated with an AIK not owned by ITadmin.

In both cases described above ITadmin and not Bob will receive the data intended for Bob. The case where ITadmin has a further TPM3 for his own platform PF3 available does not make any difference in this attack scenario.

6 Risk mitigation in the further evolution of the TPM specification

As explained in the introduction, the security problems introduced here are the result of work conducted for the German Federal Office for Information Security (BSI). We introduced the results of our work into the TPM working group of the TCG (the ones explained in this paper and others as well). In consequence newer revisions of the TPM specification include remarks that point out the encountered problems and and give hints as to how they can be avoided.

In order to avoid the possibility to manipulate the TPM_CertifyKey command a so-called *transport session* can be established that encrypts command and response data. To do this, before starting the command sequence shown in Figure 1, the TPM user that wants the TPM to generate a certificate for a key has to do the following (for example directly before sending the TPM_CertifyKey command, see Figure 1).

- start an OIAP session with TPM_OIAP and load a key she owns with TPM_LoadKey
- use the public part of this key to encrypt a symmetric key to be used within the transport session (inside or outside of the TPM),
- set the TPM_TRANSPORT_ENCRYPT flag within TPM_TRANSPORT_ATTRIBUTES which is contained in TPM_TRANSPORT_PUBLIC, one of the parameters of the TPM command,
- send TPM_EstablishTransport to the TPM containing TPM_TRANSPORT_PUBLIC and the encrypted session key.

The TPM generates a transport session handle, decrypts the session key and stores both for later use (note that for transport sessions as well as for OIAP and OSAP sessions, a so-called rolling nonce procedure is used, we omit the technical details here). It then returns the transport handle (among other information).

Now instead of sending TPM_CertifyKey in clear, the command has to be wrapped with the established session key and included as *wrappedCmd* parameter in TPM_ExecuteTransport, refering to the transport session using the respective handle. The TPM decrypts the command, processes it as usual and returns the wrapped response. Note that wrapping the command means that encryption is performed only on the command data, i.e. key handles and HMACs etc. are kept in cleartext. Hence this does not yet prohibit to manipulate TPM_CertifyKey. However, if the command parameter TPM_NONCE that can be used to ensure the freshness of the certificate indeed contains a nonce which is additionally not known to a possible attacker, she can exchange the key handle but can not produce the respective HMAC since this has to cover the nonce as well. So confidentiality of the nonce is an important precondition for prohibiting manipulation of TPM_CertifyKey using a transport session.

Preserving the integrity of key blobs is more difficult and depends on the actual context the key shall be used in. In our scenario where the key is not used by the TPM but only certified, one possibility to verify the key's integrity is to load the key and then use it, for example for verifying a signature generated with this key. In cases where the key shall actually be used by the TPM different key usage authorization data will reveal that the wrong key blob has been loaded.

7 Conclusions

The Trusted Platform Module TPM, used accurately, provides the means to considerably enhance the security of platforms and systems. Yet it is highly complex and hence the specification is difficult to understand. Previous work

on the security evaluation of TPM functionality has covered only small details. In this paper we presented some results of a first systematic security evaluation covering large parts of the specification. Our work revealed certain problems that can lead to security flaws and resulted in later revisions of the specification containing remarks that address these problems. Work by the TPM working group of the TCG on the specification is still in progress, our future work will address security evaluations of next generation TPM specifications.

Acknowledgement

We thank the members of the TPM working group of the TCG, in particular David Grawrock and Ken Goldman, for fruitful discussions and their kind support regarding our collaboration. We further thank Thomas Caspers for helpful suggestions.

References

1. TCG Trusted Computing Group. TPM Main Part 2 TPM Structures Specification Version 1.2 Level 2 Revision 103. www.trustedcomputing.org, 2007.
2. TCG Trusted Computing Group. TPM Main Part 3 Commands Specification Version 1.2 Level 2 Revision 103. www.trustedcomputing.org, March 2007.
3. Trusted Computing Group. TCG TPM Specification 1.2. www.trustedcomputing.org, 2006.
4. S. Gürgens, P. Ochsenschläger, and C. Rudolph. Authenticity and Provability – a Formal Framework. GMD Report 150, GMD – Forschungszentrum Informationstechnik GmbH, 2001.
5. S. Gürgens, P. Ochsenschläger, and C. Rudolph. Role based specification and security analysis of cryptographic protocols using asynchronous product automata. In *DEXA 2002 International Workshop on Trust and Privacy in Digital Business*. IEEE, 2002.
6. S. Gürgens and C. Rudolph. Security Analysis of (Un-) Fair Non-repudiation Protocols. *Formal aspects of computing*, 2004.
7. R. Needham and M. Schroeder. Using encryption for authentication in large networks of computers. *Communications of the ACM*, 21:993–999, 1978.
8. P. Ochsenschläger, J. Repp, and R. Rieke. Abstraction and composition – a verification method for co-operating systems. *Journal of Experimental and Theoretical Artificial Intelligence*, 12:447–459, 2000.
9. P. Ochsenschläger, J. Repp, R. Rieke, and U. Nitsche. The SH-Verification Tool – Abstraction-Based Verification of Co-operating Systems. *Formal Aspects of Computing, The Int. Journal of Formal Methods*, 11:1–24, 1999.
10. D. Otway and O. Rees. Efficient and timely mutual authentication. *Operating Systems Reviews*, 21:8–10, 1987.
11. R. Sailer, X. Zhang, T. Jaeger, and L. van Doorn. Design and implementation of a TCG-based integrity measurement architecture. In *Proceedings of the 13th USENIX Security Symposium*. USENIX Association, 2004.