Property-based Attestation for Computing Platforms: Caring about properties, not mechanisms*

Ahmad-Reza Sadeghi
Ruhr-University Bochum, Germany
sadeghi@crypto.rub.de

Christian Stüble
Ruhr-University Bochum, Germany
stueble@acm.org

ABSTRACT

Over the past years, the computing industry has started various initiatives announced to increase computer security by means of new hardware architectures. The most notable effort is the Trusted Computing Group (TCG) and the Next-Generation Secure Computing Base (NGSCB). This technology offers useful new functionalities as the possibility to verify the integrity of a platform (attestation) or binding quantities on a specific platform (sealing).

In this paper, we point out the deficiencies of the attestation and sealing functionalities proposed by the existing specification of the TCG: we show that these mechanisms can be misused to discriminate certain platforms, i.e., their operating systems and consequently the corresponding vendors. A particular problem in this context is that of managing the multitude of possible configurations. Moreover, we highlight other shortcomings related to the attestation, namely system updates and backup. Clearly, the consequences caused by these problems lead to an unsatisfactory situation both for the private and business branch, and to an unbalanced market when such platforms are in wide use.

To overcome these problems generally, we propose a completely new approach: the attestation of a platform should not depend on the specific software or/and hardware (configuration) as it is today's practice but only on the "properties" that the platform offers. Thus, a property-based attestation should only verify whether these properties are sufficient to fulfill certain (security) requirements of the party who asks for attestation. We propose and discuss a variety of solutions based on the existing Trusted Computing (TC) functionality. We also demonstrate, how a property-based attestation protocol can be realized based on the existing TC hardware such as a Trusted Platform Module (TPM).

*This research work has been done within the European project ECRYPT.

NSPW 2004 Nova Scotia Canada

1. INTRODUCTION

The rapid expansion of world-wide connectivity has changed the requirements on IT systems. We require systems which can guarantee authenticity, integrity, privacy, anonymity, and availability. Cryptography and many other technical security measures such as firewalls, Intrusion Detection Systems and so on are useful tools providing solutions to a variety of security related problems. However, they would work properly only if the underlying computing platform, in particular the operating system, is secure. Existing computing platforms, however, suffer under various security problems due to their architectural weaknesses in hardware and software as well as their complexity [30].

In this context the computing industry has come up with Trusted Computing (TC), a new generation of computing platforms based on new architectures both in hardware and software. The results of their investigations are the two well-known initiatives by the TCG (Trusted Computing Group)[1], an alliance of leading IT enterprises, and Microsoft's NGSCB (Next-Generation Secure Computing Base)[2]. Whereas there is no technical specification for NGSCB available yet, TCG has published the corresponding hardware specifications [34, 14].

The stated goal of these architectures is to improve the security and trustworthiness of computing platforms [20, 21, 30, 29]. Indeed, these platforms offer many useful functions which can be used to increase a platform's security. They extend the conventional PC architecture by new mechanisms to (i) protect cryptographic keys, (ii) generate random numbers in hardware, (iii) authenticate (the configuration of) a platform (attestation), and (iv) cryptographically bind the data to be encrypted to certain information, e.g., the system configuration and the identifier of the invoking application (sealing).

However, there is still an ongoing public debate about the negative economical, social, and technical consequences of these platforms [2, 4, 33]. People are concerned about the potential dangers that can be caused by the capabilities of such platforms: they may give vendors and content providers too much control over personal systems and users' private information. Although most complains about trusted computing are speculative[3], it is highly important to observe

[1] www.trustedcomputinggroup.org
[2] www.microsoft.com/ngscb/
[3] As pointed out by [27] the TC functionalities are com-

its development carefully, and to improve this technology by putting together the missing pieces for more secure platforms in future.

In this paper we consider the deficiencies of TCG regarding the two important functionalities attestation and sealing. We show that the existing proposals for their realization allow a remote instance to discriminate certain platforms having certain configurations (of the hardware and software running on the platform). Thus, a remote instance, attesting a platform, is able to exclude certain configurations from his/her business model, e.g., configurations related to alternative operating systems such as Linux. Hence, if in the future attestation is used to enforce the software configuration, powerful vendors may enforce their policies on their products by preventing alternative software products from running on their platforms.[4] Further, the existing proposals enable an instance attesting a platform or observing the attestation to obtain complete information about the hardware and software configuration[5] of a platform making attacks on a platform easier.

Other problems related to attestation are updates and backup: the new functionalities allow to seal critical data (e.g., documents, content) to a certain platform configuration. This, however, strongly limits the usage flexibility when system updates (e.g., patches) change the system configuration. As a consequence, the data is not accessible anymore. Similar situations arise when a system backup is made. It is not possible to access the data on another platform having a different configuration (even if this platform satisfies the same requirements as the previous one). Thus, also the sealing function can be applied to limit or prevent the use of certain products.

Clearly, the above mentioned problems lead to an unsatisfactory situation which is not what we understand under security for *all* involved parties (in the sense of multilateral security) and under an open market.

Our proposal is therfore to pursue a completely different and new approach which uses these functionalities only based on the properties a platform offers and not based on the configuration of its software and hardware components. A property of a platform describes an aspect of the behavior of that platform regarding certain requirements, such as security-related requirements (e.g., that a platform has built-in measures in accordance to the privacy laws). Hence, different platforms with different components may have different configurations while they may all offer the same properties and consequently fulfill the same requirements.

We propose and discuss several solutions to the mentioned problems following the property-based attestation paradigm. Our solutions differ in their trust models, efficiency and the functionality offered by the trusted components. Our proposals can be applied to all approaches that provide some kind of secure booting or application authentication. We also demonstrate, how the Trusted Software Stack (TSS), the TPM-library proposed by the TCG, can be extended by a property-based attestation protocol based on the existing TC hardware[6] without a need to change the underlying trust model.[7]

2. CONVENTIONS

In this section we introduce the basic terminology and definitions used throughout this paper.

- *Roles*: The instance who is interested in authenticating a platform (the output of the attestation) is called *challenger* (e.g., a local user or a remote instance such as a service or content provider). The platform, to be authenticated, measures (determines) and attests its configuration (see Section 3 for an example), and is called *attestor*. The attestor is used by *users* (e.g., a local user or platform owner such as a company). A trusted third party is denoted by TTP. Note that by "trust" we mean the assumption that TTP does not violate a certain security requirement. The abbreviation TC stands for *Trusted Computing* as a general term meaning the technology which provides trusted components on a platform.

- *Cryptographic primitives*: We denote an asymmetric encryption scheme with the tuple $(\mathsf{GenKey}(), \mathsf{Enc}(), \mathsf{Dec}())$ for key generation, encryption and decryption algorithms. (pk_X, sk_X) denotes the public and secret key of a party X. We sometimes denote the encryption and decryption keys of an instance X with ek_X and dk_X.

  Further, a digital signature is denoted by a tuple $(\mathsf{GenKey}(), \mathsf{Sign}(), \mathsf{Verify}())$ for key generation, signing and verification algorithms. With $\sigma \leftarrow \mathsf{Sign}(sk_X; m)$ we mean the signature on a message m signed by the signing key sk_X. The return value of the verification algorithm $ind \leftarrow \mathsf{Verify}(pk_X; m, \sigma)$ is a Boolean value $ind \in \{true, false\}$. A certificate on a quantity Q with respect to a verification key pk_X is denoted by $cert(sk_X; Q)$, a signature generated by applying the corresponding signing key.

pletely under the control of the underlying operating system, and as a consequence, users can benefit from the security features of TCG/NGSCB, as long as their operating system is trustworthy.

[4]Note that similar approaches can be observed today, e.g., many banks provide banking software for only one operating system, a lot of websites support only one web-browser, and a lot of hardware devices, like music players, expect a specific operating system. However, in conjunction with TC technology, the use of alternatives will become impossible.

[5]One may think that the subject of attestation is only to determine a "known" configuration. However, if this would be the case then only a binary confirmation would suffice indicating that a known configuration has been changed. This solution would be much more simpler than the attestation procedure specified by the TCG.

[6]called Trusted Platform Module (TPM).

[7]It should be noted that our primary goal is to have a non-discriminating attestation as a standard, which can be certified by trusted entities, and on which the vendors and developers of related products should rely. Clearly, standards leave some space for corresponding implementations, and this may open the door for information flow allowing, e.g., operating system footprinting (see, e.g., www.insecure.org/nmap). However, this is not the subject of this paper.

- *Trust model*: Systems (entities or components) that can violate the security requirements of both attestor and challenger are denoted as *fully trusted*. An example of a fully trusted component is the Trusted Platform Module (TPM) explained in Section 3. Systems that can only violate the security requirement of one party are denoted as *trusted by the attestor* or *trusted by the challenger*. The Trusted Software Stack (TSS) [13] defined by the TCG is an example of a component that only has to be trusted by the attestor. All other systems are *untrusted*.

3. TC OVERVIEW

The Trusted Computing Group (TCG) has published a specification [34, 14] of hardware components that, if fully trusted, provides security functions required by secure operating systems for trustworthy operations. Loosely speaking, the basic idea is to embed a "trusted third party" into the underlying hardware where this party is realized by tamper-resistant hardware components. The intention is, however, to keep the tamper-resistance assumption as weak as possible and reduce the cost by keeping the trusted component as small as possible. Beside a secure random number generator and tamper-resistant storage, the specification defines a mechanism, called *attestation*, that attests the platform configuration (e.g., of the BIOS and the TCB) that was determined by a cryptographically secure hash function SHA-1 [17].

For better understanding we take look at an example: on platform startup a hardware component, called the Core Root of Trust module (CRTM), hashes the current BIOS and the master boot record (MBR) of the boot device and writes the result into protected registers called Platform Configuration Registers (PCR) which are located in an (tamper-resistant) hardware module called Trusted Platform Module (TPM). The software that is executed in the MBR (e.g., the boot loader) can create a chain of trust by writing a hash value of the software it loads (e.g., the operating system) into another PCR register. We hence call the chain of hash values stored in the PCR as the *platform configuration*.[8] The TPM can cryptographically sign the current platform configuration using a protected signature key, to attest it to a remote challenger.

To ensure that data sent to a platform can only be accessed under a specific platform and platform configuration, the *sealing* mechanism is provided that binds data to a platform configuration: a remote instance can encrypt critical data including a demanded configuration under an encryption key whose decryption key is only known to the TPM. The TPM can decrypt the cipher but releases the data only if the current platform configuration matches the demanded one. Note that sealing functionality can be used to ensure that the critical data cannot be accessed by the user when the configuration of the platform changes after the attestation took place.

Microsoft's NGSCB [20, 21] is one concrete instantiation of the TCG specification based on additional hardware extensions such as LaGrande[9] Since the NGSCB/LaGrande approach is based on the functionality provided by the TCG specification, we concentrate in the following on TCG, although our solutions are applicable to NGSCB, or any other approach that offers the same functionalities.

The public debates of the past highlighted several deficiencies and unsatisfying properties of the current TCG specification version 1.1b [34]. In October 2003, the TCG published an updated TCG specification version 1.2 [14] which solves some of these deficiencies. For instance, a cryptographic protocol called *Direct Anonymous Attestation* (DAA) [7] was specified that, roughly spoken, provides users with an unlimited number of pseudonyms without requiring a trusted Privacy Certification Authority (privacy-CA) that was required (and criticized) in version 1.1b of the specification. Note that the anonymity provided by DAA or privacy-CA's is completely orthogonal to the stated goals of this publication. Nevertheless, we will sketch in Section 5 how both approaches can be combined into an anonymous property-based attestation function.

3.1 TCG Model

The abstract model for the basic functions provided by TCG-compliant platforms is illustrated in Figure 1: it consists of two state transition machines. The first one U represents the untrusted platform, while the other, fully trusted TCG, represents the TPM and the CRTM. The third machine C represents a challenger.

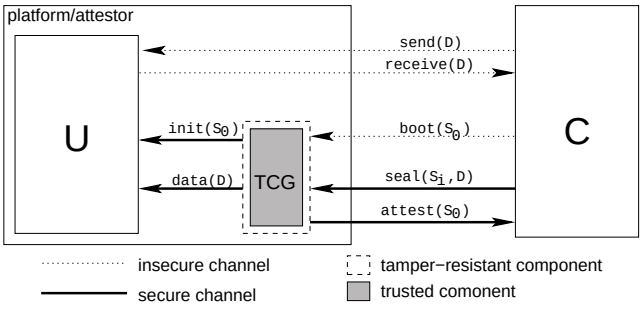

Figure 1: Abstract Model of the TCG functionality

The machines are connected by insecure and secure (authentic, integer and confidential) directed communication channels: C can communicate with U using an insecure input channel *send*() and an insecure output channel *receive*() which are used for normal communication. TCG provides an insecure initialization channel *boot*() to the outside world (e.g., to C). It accepts an initial state S_0 of U which is locally stored and then forwarded to U using the secure channel *init*(). Compared to the boot example discussed in Section 3, S_0 represents the hashed chain of the basic modules (i.e., BIOS, boot-loader, operating system) at the time the operating system is bootstrapped.

A secure output channel *attest*() returns to C the state used to initialize U. A secure input channel of TCG called *seal*() receives data D and a demanded configuration S_i. If S_0 is

[8]Note that here we do not mean the hash value of the history but rather the hash of the TCB (trusted computing base) state that remains unchanged during the run-time in contrast to, e.g., history measurements done in [31].

[9]see http://www.intel.com/technology/security/index.htm

equal to S_i, TCG sends D to U using the secure output channel $data()$.

3.1.1 TCG Assumptions

This functionality mentioned above is provided under the following conditions:

1. The platform configuration cannot be overwritten after measurements, i.e., after the hash values are computed and securely stored in TPM. Since TCG makes only statements about the initial state of U, it is important that U's state cannot be maliciously overwritten after startup.
2. Given a valid set of hash values, the challenger can determine whether the platform configuration S_0 is trustworthy.
3. TCG realizes the secure channels using two different measures: the channels between TCG and U are assumed to be secure since both components are integrated on one hardware. The other communication channels are secured based on a public key infrastructure (PKI) [14].

The first condition is hard to achieve since currently available operating systems can easily be manipulated, e.g., by exploiting security bugs or by changing memory which has been swapped to a harddisk. The same holds for the second condition since it is very difficult, if not impossible, to define the trustworthiness of today's complex platforms including all applications. Hence, without the appropriate secure operating systems these assumptions are too strong to build secure systems upon. However, a secure operating system can be efficiently provided by security kernels based on micro-kernel architectures [25, 26, 28, 10]. We will consider this in more detail in Section 5.2.2.

3.2 Deficiencies of TCG Attestation

While the attestation and the sealing mechanisms provided by TCG allow many meaningful applications (see, e.g., [30, 11, 18, 32]), the use of the platform configuration as a basis for determining its trustworthiness has some important drawbacks:

- If a challenger (e.g., providers of digital content) wants to enforce its access control rules on a platform (e.g., consumer platform) it has, in theory, to analyze whether every existing operating system and every patch-level has the "desired" configuration. Since in practice it is hard to verify the trustworthiness of every platform configuration, a challenger is forced to focus on a few predefined (mainstream) configurations. This, however, has the potential danger that "alternative" software products (e.g., OpenOffice[10] or WINE[11] and operating systems such as Linux) get isolated, and it would be more difficult (if not impossible) for them to become established on the market. One can imagine a situation where global players such as content providers and large operating systems vendors collaborate, and exclude specific operating systems and the corresponding applications. This is an undesired situation providing the ground for building monopoly, and is not promotive for the market.
- Further, the recipient of the attestation protocol or an observer gets exact information about the hard- and software configuration of a specific platform. This makes attacks to such platforms much easier, since an adversary does not need to perform more complicated platform analysis.
- Since the sealing mechanism provided by the TCG-hardware binds encrypted content to a specific system configuration, system updates, e.g., patches, that would lead to changes to the PCR values make the encrypted content unaccessible.
- Sealing can also have negative consequences, since application vendors can bind the application data to their application, making it impossible for alternative software products to be compatible. With TCG, e.g., a vendor could prevent that OpenOffice can read Word documents.

[10] www.openoffice.org
[11] www.winehq.org

4. PROPERTY-BASED ATTESTATION

The attestation and sealing function provided by TCG-compliant hardware attests the system configuration of the consumer platform that was determined on system startup. For (nearly) all practical applications, the challenger is not really interested in the specific system configuration. As we have argued in Section 3.2, this has even some disadvantages due to the multitude of possible configurations a challenger has to manage. In fact the challenger is interested in whether the attested platform provides the desired "properties". With *platform property* P we informally mean a quantity that describes an aspect of the behavior of that platform with respect to certain requirements, e.g., a security-related requirement. In general, platform properties of different abstraction levels are imaginable: A platform property may, e.g., state that a platform has built-in measures conform to the privacy laws, or that it strictly separates processes from each other, or that a platform has built-in functionalities to provide Multi-Level Security (MLS) and so on. The question of whether there is a correct or useful property set depends heavily on the underlying application and its requirements on the environment. For instance a useful property is what is now accepted as secure operating system providing isolation of processes or confinement etc.

It is out of the scope of this paper to discuss how platform properties can be evaluated. Nevertheless, a similar approach like in the context of Common Criteria Protection Profile evaluations is imaginable: Demanded platform properties can, for instance, be summarized in an implementation independent *Property Profile*.[12] Trusted third parties can certify that a platform of a specific configuration provides this properties after a successful platform evaluation. Note that using property-based attestation there will be no

[12] A possible property profile, e.g., published by content providers, could summarize all requirements of a digital rights management platform.

need to force users to deploy products of original manufacturer.[13]

Therefore, for privacy aspects and for practical reasons, an attestation mechanism is desirable that attests the properties of platforms independent of the concrete hard- and software configuration. We say that a configuration C *provides* the property P. Further, we say that a property P_i *is compatible* to another property P_j if P_i has the same security-related behavior as P_j.

4.1 Ideal TC Component

An ideal TC component is capable of determining the properties $\{P_0 \dots P_n\}$ provided by a system configuration and to decide whether a given system configuration S_0 provides a specific property P_i. Therefore, it can perform a *property-based attestation* and *property-based sealing* mechanisms as shown in Figure 2.

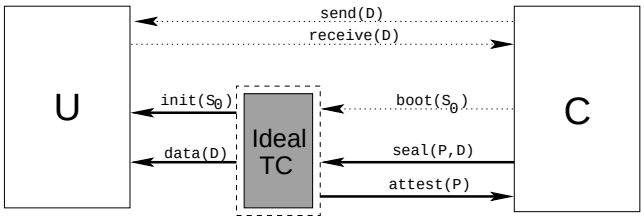

Figure 2: The ideal model of a property-based attestation function.

An ideal property-based attestation mechanism fulfills the following requirements:

1. *Security*: The ideal TC only attests certified platform configurations.
2. *Accountability*: Although the TTP has to be trusted by all participants, it is desirable to be able to detect TTP's misbehavior.
3. *Revokeability*: The TTP should be able to selectively revoke TCG versions and TCG realizations under certain circumstances, e.g., if TPM has been broken.
4. *Non-Discrimination*: Challengers should neither be able to favor selected configurations nor should they get information about the configuration of the user platform.
5. *Unlinkability*: Challengers should not be able to link two different attestation sessions.
6. *Availability*: When modifying a platform configuration without changing the provided properties, access to sealed data should be possible.
7. *Privacy*: Users should be able to control which property their platforms attest.
8. *Reduced Complexity*: Security enhancements to U should not be costly. Thus, the complexity of potentially required TCG-extensions should be low.[14]

Unfortunately, in practice it is difficult, if not impossible, to determine or compare properties enforced by a platform configuration, as it is required to follow the property-based attestation paradigm. For the long term, proof-carrying code [23, 22, 24, 3] may give the opportunity to build a TC component that determines some specific properties provided by a specific platform configuration. Today, not even content providers are able to formally specify the demanded properties, as it would be necessary to use proof-carrying code or formal analysis.

4.2 Real World TC Component

To overcome the problems that occur in the context of the current attestation and sealing implementations, we will discuss in the following various real world solutions. We first introduce the basic idea, where a trusted third party TTP is involved that transforms platform configurations into properties and vice versa. In Section 5 we vary the basic model with respect to several parameters such as the trust model, complexity, costs and functionality resulting in various solutions: Some solutions require the trusted components in hardware to be extended under realistic assumptions. In particular, we propose solutions which do *not* require any extension to the trusted hardware component and the underlying trust model.

The basic idea is to extend the ideal model with a TTP who attests that a given platform configuration S_0 fulfills a demanded property P. Thus, we replace the automatic property derivation based on configurations, required for the solution with ideal TC, by an off-line certificate issued by a TTP (see Figure 3). The certificate of TTP attesting that a configuration S_0 provides a property P is called *property certificate* and denoted by $cert_{TTP} := cert(sk_{TTP}, S_0, P)$. Note that TTP confirms the correctness of the correspondence between the platform configuration and certain properties according to defined criteria. However, as it is common practice such organizations are only liable for intentional misbehaviour and not for undetected weaknesses (compare with safety and security tests or common criteria). As mentioned before the TTP is fully trusted (i.e., by the attestor and the challenger), since both have to assume that the TTP certifies only configurations that really enforce the attested property. The extended TC component, denoted by (TC+), internally translates configurations S_i into properties P_j (and vice versa) to perform the attesting and sealing functions.

To prevent a flood of desired properties, the involved parties can, e.g., together define earmarked property profiles. For instance, end-users could define a privacy-protecting Common Criteria [9] protection profile, while content providers define a content-protecting profile. The TTP then certifies whether given configurations are compatible to that protection profiles. If the TTP is a governmental authority it can also analyze whether a given platform configuration protects

[13] For instance in some countries the deployment of car components from original manufacturers is even supported by the law. This, however, disables smaller companies from selling comparable products.

[14] One objective of TCG was to keep the these enhancement as cheap as possible

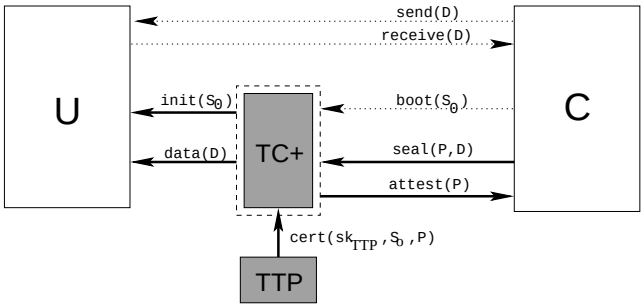

Figure 3: Real world model of property-based attestation: A trusted third party (TTP) translates demanded properties into concrete platform configurations and issues property certificates.

the consumer's privacy, e.g., by certifying that it is compatible to privacy laws.

Our solutions are all based on the current TCG specification, because it is today the mostly known and available extension of conventional computer systems that provides the measurement and attestation of integrity metrics. In general, our property-based attestation paradigm can be applied to all mechanisms that provide some kind of integrity protection (e.g., see [35, 5, 15]).

5. REALIZATION

In this section we discuss various solutions for the property-based attestation (and sealing). Some solutions require an extension of the existing TC hardware (Section 5.1) while others propose to implement the required functionalities by a fully trusted software where we analyze to what extend and under which assumptions such a software service can provide these features (Section 5.2.1). We also outline a solution that can reuse the existing TPM implementation but keeps the same trust model as the current TCG specification (Section 5.3).

5.1 Extending TCG Hardware

Extending the TPM by property-based functionality has two major advantages compared to other solutions:

- The trust model related to the trusted computing platform does not change, since the TPM has to be fully trusted anyway. Thus, the requirements 1 (security), 2 (accountability) and 4 (non-discrimination) can be fulfilled.
- Since the realization of property-based attestation within a TPM does not depend on external components, changes to the platform configuration cannot lead to unavailability of sealed data (see requirement 6).

The disadvantage of this approach is the additional complexity of the TPM which makes the TPM more expensive. Nevertheless, the required complexity should be acceptable compared to the complexity of the DAA protocol [7].

5.1.1 Property-based Attestation Using Certificates

An intuitive solution is to extend the TPM hardware in such a way that it checks whether a valid property certificate exists that was created by a party that is trusted by the challenger. The realization of the property-based attestation is quite simple: the TPM receives the demanded property P, a public key of a TTP that is trusted by the challenger pk_{TTP}, a property certificate $cert(sk_{TTP}; P', S')$ and nonce r for freshness. The TPM verifies the property certificate and whether $P' = P$ respectively whether $S' = S_0$. If positive, it generates and returns a property-based attestation certificate $cert_{TPM} := cert(sk_{TPM}; P, r)$ where the corresponding verification key pk_{TPM} can be seen as a pseudonym of TPM.

Fulfilling requirement 5 (unlinkability) should not be problematic, since CA-based pseudonyms, or the DAA protocol, can be used to blind the signature key sk_{TPM}.

The use of certificates that guarantee platform properties has the commonly known problems of public key infrastructures, especially in the context of certificate revocations (e.g., if a new bug is becoming public that undermines a certified property). Allowing the TTP to revoke a property certificate (requirement 3) thus requires additional TPM support ensuring that challengers can recognize if a revoked property certificate was used. A simple but unsatisfying solution is the use of (short) validity periods of property certificates.

5.1.2 Group Signatures

A group signature scheme allows a group member to sign messages anonymously on behalf of the group. In the case of a dispute, the identity of a signature's originator can be revealed (only) by a designated entity [8]. In our model, the public group signature key represents a property P, while the appropriate private group signature keys generated by the TTP represent different configurations providing the same property P. Since the verifier of a group signature cannot decide which secret key has been used to generate the signature, it does not get information about the configuration of the attestor, fulfilling requirements 4 (non-discrimination) and 5 (unlinkability).

To ensure that users cannot misuse the signature key (requirement 1, e.g., to attest a wrong configuration, the TTP binds the group signature key to the certified configuration. Some group signature schemes [6] allow a designated entity to exclude group members, thus selected (e.g., insecure) configurations can be revoked (requirement 3). They also allow the designated entity to add new group members and thus to add new compatible configurations (requirement 6). Users can decide which group signature key they load onto their platform, therefore also the privacy requirement is fulfilled. Appendix B outlines how the initialization, attestation, and sealing protocols can be realized.

5.2 Extending TCG Software

5.2.1 Using Trusted Attestation Service (TAS)

If it is demanded not to modify the TPM, the extensions discussed in Section 5.1 can be realized as separated trusted third party that translates attestation requests into configurations online. Since online services are often performance

critical, we analyze in the following sections to what extend TCG functionality allows us to implement a trusted attestation service (TAS) as distributed (fully) trusted software service that extends the functions provided by a TPM, and that is executed on top of the operating system of the attestor (platform) (see Figure 4).

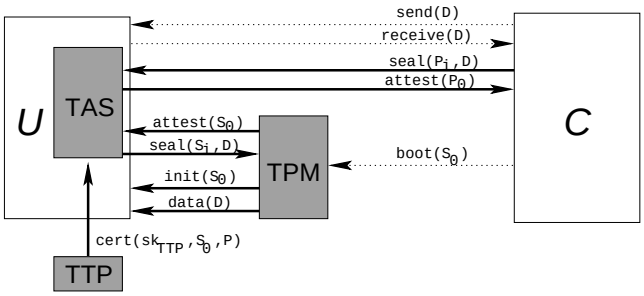

Figure 4: Property-based functions can be provided by a fully trusted software service (TAS) that is executed on the user's platform.

Both, TAS and operating system have to be fully trusted: this opens new demands on the underlying software architecture. How these requirements can be fulfilled is discussed in the following section.

5.2.2 The Basic Architecture

The maximum level of trustworthiness of a TCG platform is limited by the trustworthiness of the TCB components. Today's operating systems are definitely inappropriate to be used as a trusted software basis that cannot be manipulated after it was booted[15]: they have conceptual security flaws (e.g., an administrator who can bypass every security enforcement mechanism[16]) and error prone implementations resulting from high complexity[17].

Therefore, we assume in the following that the platform contains a very small TCB based on a minimal security kernel which is capable of protecting the TCB from other (untrusted) software. Example instantiations of such an architecture are Microsoft's Next-Generation Secure Computing Base (NGSCB) [10], which has a kernel that is called *nexus*, or the PERSEUS security architecture [25, 26, 28] that uses a microkernel of the L4-family [19] as its basis (see Figure 5).

In practice, it is very difficult for a challenger to decide whether a concrete system configuration provides a desired property. Even if the enforcement mechanisms of the trusted computing base would be highly trustworthy (e.g., because it was evaluated at EAL7[18]), the property obviously depends on the locally enforced security policy, too. To make the analysis of the platform's trustworthiness more realistic, it

[15] Note that the operating system has no access to the cryptographic keys stored in TPM, but to all decrypted contents.
[16] The administrator can, for instance, at any time change the state of the operating system.
[17] By exploiting a bug, e.g., based on a buffer overflow, an attacker can control the operating system kernel, and thus, change its state.
[18] Evaluation Assurance Level, see [9]

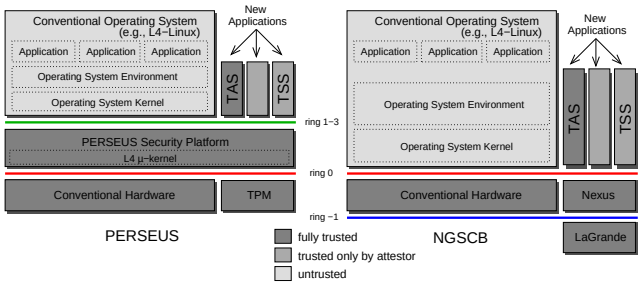

Figure 5: Possible realizations of the basic trustworthy architecture that provides trusted services and applications.

is in our opinion meaningful to provide a policy-neutral operating system base that delegates the enforcement of policies to the application level software (see [28]). Since the underlying TCB is now much more simpler, it has only to fulfill simpler security requirements.

- *Secure Path*: The TCB has to provide a secure path between provider and application, e.g., it has to ensure that only the provider's application can access the unsealed content. This requirement ensures that the challenger and the TAS can communicate securely.
- *Protected Domain*: The TCB has to prevent that an attacker (e.g., the local user or a concurrent application) can access or manipulate the code or the data of the application (which is similar to overwriting S_0). This requirement ensures that the code and the data of the TAS are protected against attacks of concurrent processes.

Both security architectures, PERSEUS and NGSCB, fulfill these requirements [25, 26, 10].

Due to the fact that the module which realizes the property-based functions is not a tamper-resistant module, some functional limitations and changes to the trust model arise:

- Note that as mentioned in Section 3.1.1 the level of the trust that can be attested is limited by the trustworthiness of the underlying security kernel.[19] The maximum trustworthiness of the TAS/OS pair has to be considered by the TTP that certifies the properties that are provided by a configuration.
- Since the policy enforced by the TAS obviously depends on the security kernel and its own implementation, requirement 6 (availability) can only be fulfilled in a limited way, since changes to one of the two components leads to inaccessibility to the sealed data. Therefore, the TAS has to provide a secure update

[19] For instance, if the TAS is implemented based on Linux or Windows, it does not make sense to attest configurations that are more trustworthy as these operating systems, since an attacker can control the TAS by hacking the operating system.

function that seals protected keys under the new configuration before changing to that configuration (see Section 5.4).

- The integrity of the TAS has to be ensured by binding security-critical data to a platform configuration that includes a correct TAS.
- While the TCG specification requires some kind of tamper-resistance of the hardware modules, software extensions can easier be manipulated, e.g., by eavesdropping the memory bus.

5.3 Solutions Without a Trusted Service

The trust model or the limited functionality of the software-based realization of the property-based attestation may not be satisfying. In the following we sketch a protocol that can securely prove to the challenger that the platform configuration provides a required property.

Since the challenger can verify the proof it does not need to trust the implementation of the software service module anymore. Only the user of the platform has to trust that the software performing the protocol does not leak information about the platform configuration. Software under this trust model is summarized by the TCG as the Trusted Software Stack (TSS) [13]. Thus, the implementation of the following protocol can be seen as an extension to the TSS.

5.3.1 Proving Possession of Valid Property Certificate

The suggested property-based protocol is based on the existence of valid property certificates as described in Section 5.1.1.

The basic idea is that the platform proves that there is a valid link between conventional attestation signature $sig_{TPM} := \mathsf{Sign}(sk_{TPM}; S_0, r')$ created by the TPM and the certificate $sig_{TTP} := \mathsf{Sign}(sk_{TTP}; S_0', P')$ attesting that configuration S_0 provides P. Here r' denotes a nonce.

Given the (common) inputs the verification key of the TPM pk_{TPM}, the verification key pk_{TTP} of a TTP that is accepted by the challenger, the desired property P and a nonce r. The protocol should output TRUE to the challenger C, if the following holds:

- TPM's attestation signature is valid, i.e.,

$$true \stackrel{?}{=} \mathsf{Verify}(pk_{TPM}; sig_{TPM}),$$

- the property certificate is valid, i.e.,

$$true \stackrel{?}{=} \mathsf{Verify}(pk_{TTP}; sig_{TTP}),$$

- the desired and the certified properties are the same, i.e., $P' = P$,
- the certified and the attested configurations are the same, i.e., $S_0' = S_0$, and
- the nonce r entered by the challenger C and the nonce signed by the TPM are equal, i.e., $r' = r$.

The above statements can be proven by general zero-knowledge proofs since they are NP statements [12]. More concretely, one can prove that he/she has a signature/certificate without disclosing how the signature/certificate looks like. Note that the signatures sig_{TPM} and sig_{TTP} are the secret inputs to the protocol since the challenger should not learn information about the platform configuration S_0. For efficient construction of such protocols one may use cryptographic techniques similar to those used in the context of DAA.[20]

The unlinkability requirement (requirement 5) can be realized by using sk_{TPM} as an anonymous session signature key that was verified by the challenger using a CA-based or DAA-based pseudonyms.

5.3.2 Proof of membership

A configuration blinding mechanism which was suggested in [16] is to encrypt S_0 values and sign a contract between provider and consumer that defines the property that the provider and the consumer agree on. To prevent that the consumer from cheating, only S_0 values encrypted under the public key of a trusted third party can be transmitted. This approach makes it impossible for providers to discriminate an operating system or to force users to use a specific platform configuration. Since the TPM also signs the encrypted values, providers can be sure that the attested configuration is valid. In case of a conflict, e.g., if a user accuses the bank, the TPM vendor (or the court) can decrypt these values to verify whether the user used the correct banking software or another one. The problem of this approach is that providers cannot verify the property enforced by the user platform online. In this section we discuss an improved method that allows online verification.

Instead of using a court to decrypt the configuration offline, the TPM can also cryptographically prove that the blinded configuration is contained in a set of certified configurations. To use a *proof of membership* protocol for property-based attestation, the TTP publishes a list of all platform configurations $S_1 \ldots S_n$ that provide a specific property P. To attest properties, a local software service performs a conventional attestation protocol with the TPM where it hides the signed configuration. Then, a cryptographic protocol proves that

- the blinded configuration value is contained in the list of configurations published by the TTP, and that
- the TPM attestation signature is valid.

Here one can apply cryptographic techniques such as commitments and zero-knowledge proofs to prove the statement.

To fulfill the revokeability and the availability requirements, the underlying proof of membership protocol should offer the possibility to dynamically remove configurations from the list respectively to add new configuration into the list.

[20] A slightly extended protocol can be used for property-based sealing: the attestor has to prove that it knows a valid certificate $\mathsf{Sign}(sk_{TPM}; ek, S_0)$ on the encryption key ek created by the TPM, and a valid property certificate $\mathsf{Sign}(sk_{TTP}; S_0', P')$.

To fulfill the non-discrimination requirement completely, the underlying protocol should allow the TTP to keep the list of configurations secret. Otherwise, a remote challenger could discriminate single configurations by accepting only those properties which do not contain them.

5.4 TCB Updates

If the realization of property-based attestation depends on the integrity of software components (as the solutions that we pointed out in this section and in Section 5.3), the availability requirement cannot be provided. The reason for this is that the current TCG specification (see Section 3) allows only to certify non-migrateable sealing keys that are bound to a fixed platform configuration. If the configuration changes, the TPM denies the use of these keys.

One solution to that problem is the extension of the TCB by update functionality: if, e.g., users can seal the private keys of the TAS under another configuration by using a certificate that states that the new configuration provides the same property as the old one, updates of the TCB become possible without violating security policies of all involved parties.

Update Example: Consider an operating system configuration S_0 that provides P certified by $cert_{TTP} := cert(sk_{TTP}, S_0, P)$. Further, the TAS uses a secret key SK that is bound to the configuration S_0. The update function allows the user of the attestor to bind SK to another configuration S_i iff the user can provide a certificate that states that S_i provides the same properties as S_0.

Another solution is to implement the complete sealing protocol including attestation of the encryption keys into the software. The trust model will not change, since the challenger has to trust the configuration which has access to the sealed data anyway. Note that the operating system cannot access the cryptographic key used for sealing directly, but it has access to the symmetric key used to encrypt the sealed content [14]. Thus, challengers can also trust that the property-based sealing protocol is trustworthy under this configuration.

6. REFERENCES

[1] ACM. *Proceedings of the 19th ACM Symposium on Operating Systems Principles (SOSP'03)*, Bolton Landing, NY, USA, Oct. 2003.

[2] R. J. Anderson. Security in open versus closed systems — the dance of Boltzmann, Coase and Moore. Technical report, Cambridge University, England, 2002.

[3] A. W. Appel and E. W. Felten. Models for security policies in proof-carrying code. Technical Report TR-636-01, Princeton University, Computer Science, Mar. 2001.

[4] W. A. Arbaugh. Improving the TCPA specification. *IEEE Computer*, pages 77–79, Aug. 2002.

[5] W. A. Arbaugh, A. D. Keromytis, D. J. Farber, and J. M. Smith. Automated recovery in a secure bootstrap process. In *Proceedings of the Symposium on Network and Distributed Systems Security (NDSS '98)*, pages 155–167, San Diego, California, Mar. 1998. Internet Society.

[6] G. Ateniese, J. Camenisch, M. Joye, and G. Tsudik. A practical and provably secure coalition-resistant group signature scheme. In M. Bellare, editor, *Advances in Cryptology – CRYPTO '2000*, volume 1880 of *Lecture Notes in Computer Science*, pages 255–270. International Association for Cryptologic Research, Springer-Verlag, Berlin Germany, 2000.

[7] J. Camenisch and E. V. Herreweghen. Design and implementation of the *idemix* anonymous credential system. In *Proceedings of the 9th ACM Conference on Computer and Communications Security*, pages 21–30, Washington, DC, USA, Nov. 2002. ACM Press.

[8] D. Chaum, , and E. van Heijst. Group signatures. In D. Davies, editor, *Advances in Cryptology – EUROCRYPT '91*, volume 547 of *Lecture Notes in Computer Science*, pages 257–265. International Association for Cryptologic Research, Springer-Verlag, Berlin Germany, 1991.

[9] Common Criteria Project Sponsoring Organisations. *Common Criteria for Information Technology Security Evaluation*, Aug. 1999. Version 2.1, adopted by ISO/IEC as ISO/IEC International Standard (IS) 15408 1-3. Available from `http://csrc.ncsl.nist.gov/cc/ccv20/ccv2list.htm`.

[10] P. England, B. Lampson, J. Manferdelli, M. Peinado, and B. Willman. A trusted open platform. *IEEE Computer*, 36(7):55–63, 2003.

[11] T. Garfinkel, B. Pfaff, J. Chow, M. Rosenblum, and D. Boneh. Terra: a virtual machine-based platform for trusted computing. In *Proceedings of the 19th ACM Symposium on Operating Systems Principles (SOSP'03)* [1], pages 193–206.

[12] O. Goldreich, S. Micali, and A. Wigderson. Proofs that yield nothing but their validity or all languages in NP have zero-knowledge proof systems. *Journal of the ACM*, 38(3):690–728, July 1991.

[13] T. C. Group. TCG software stack specification. `http://trustedcomputinggroup.org`, Aug. 2003. Version 1.1.

[14] T. C. Group. TPM main specification. `http://www.trustedcomputinggroup.org`, Nov. 2003. Version 1.2.

[15] N. Itoi, W. A. Arbaugh, S. J. Pollack, and D. M. Reeves. Personal secure booting. In V. Varadharajan and Y. Mu, editors, *Information Security and Privacy — 6th Australasian Conference, ACISP 2001*, volume 2119 of *Lecture Notes in Computer Science*, pages 130–144, Sydney, Australia, July 2001. Springer-Verlag, Berlin Germany.

[16] K. Kursawe and C. Stüble. Improving end-user security and trustworthiness of TCG platforms. Presented and basis for the panel discussion about TCG at the 33. GI-Fachtagung, Frankfurt, `http://www.prosec.rub.de/Publications/KurStu2003.pdf`.

[17] N. S. Laboratory. Secure hash standard. Federal Information Processing Standards Publication (FIPS PUB) 180-1, Apr. 1995.

[18] D. Lie, C. A. Thekkath, and M. Horowitz. Implementing an untrusted operating system on trusted hardware. In *Proceedings of the 19th ACM Symposium on Operating Systems Principles (SOSP'03)* [1], pages 178–192.

[19] J. Liedke. Improving IPC by kernel design. In *Proceedings of the Fourteenth ACM Symposium on Operating Systems Principles*, pages 175–187, Dec. 1993.

[20] Microsoft Corporation. Building a secure platform for trustworthy computing. White paper, Microsoft Corporation, Dec. 2002.

[21] C. Mundie, P. de Vries, P. Haynes, and M. Corwine. Microsoft whitepaper on trustworthy computing. Technical report, Microsoft Corporation, Oct. 2002.

[22] G. Necula. Proof-carrying code. In *24th Symposium on Principles of Programming Languages (POPL)*, pages 106–119, Paris, France, Jan. 1997. ACM Press.

[23] G. C. Necula and P. Lee. Safe kernel extensions without run-time checking. In *Proceedings of the Second Symposium on Operating Systems Design and Implementation*, pages 229–243, Seattle, Washington, Oct. 1996. USENIX Association.

[24] G. C. Necula and P. Lee. The design and implementation of a certifying compiler. In *Proceedings of the 1998 ACM SIGPLAN Conference on Prgramming Language Design and Implementation (PLDI)*, pages 333–344, 1998.

[25] B. Pfitzmann, J. Riordan, C. Stüble, M. Waidner, and A. Weber. The PERSEUS system architecture. In D. Fox, M. Köhntopp, and A. Pfitzmann, editors, *VIS 2001, Sicherheit in komplexen IT-Infrastrukturen*, DuD Fachbeiträge, pages 1–18. Vieweg Verlag, 2001.

[26] B. Pfitzmann, J. Riordan, C. Stüble, M. Waidner, and A. Weber. The PERSEUS system architecture. Technical Report RZ 3335 (#93381), IBM Research Division, Zurich Laboratory, Apr. 2001.

[27] A.-R. Sadeghi and C. Stüble. Bridging the gap between TCPA/Palladium and personal security. Technical report, Saarland University, Germany, 2003.

[28] 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.

[29] D. Safford. Clarifying misinformation on TCPA. White paper, IBM Research, Oct. 2002.

[30] D. Safford. The need for TCPA. White paper, IBM Research, Oct. 2002.

[31] R. Sailer, X. Zhang, T. Jaeger, and L. V. Doorn. Design and implementation of a TCG-based integrity measurement architecture. In *Proceedings of the 11th USENIX Security Symposium*. USENIX, Aug. 2004.

[32] R. Sailer, X. Zhang, T. Jaeger, and L. V. Doorn. Design and implementation of a TCG-based integrity measurement architecture. Research Report RC23064, IBM Research, Jan. 2004.

[33] B. Schneier. Palladium and the TCPA. `http://www.counterpane.com/crypto-gram-0208.html#1`.

[34] Trusted Computing Platform Alliance (TCPA). Main specification, Feb. 2002. Version 1.1b.

[35] J. Tygar and B. Yee. Dyad: a system using physically secure coprocessors. In *Technological Strategies for Protecting Intellectual Property in the Networked Multimedia Environment*, The Journal of the Interactive Multimedia Association Intellectual Property Project, Coalition for Networked Information, pages 121–152, MIT, Program on Digital Open High-Resolution Systems, Jan. 1994. Interactive Multimedia Association, John F. Kennedy School of Government.

APPENDIX
A. EXAMPLE TAS REALIZATION

The main task of the TAS is to verify and attest that the current platform configuration is compatible to a demanded property. The comparison is done based on certificates provided by a trusted third party (see Section 4.2), while the attestation is performed by a signature key that can only be accessed by the fully trusted TAS.

A.1 Initialization

The main task of the initialization phase is to allow users to convince external parties that the used TAS is trustworthy. For this purpose, a signature key that was generated by a TAS is certified by a TTP.

If the platform is started, it loads the security kernel and the TAS while the TPM measures the platform configuration and writes it to PCR_0 to PCR_n, while PCR_{n-1} stands for the configuration of the security kernel and PCR_n stands for the measured configuration of the TAS. Then, the other components of the TCB are loaded, followed by uncritical services and applications.

To be able to prove the trustworthiness of its TAS using a certified attestation key, the attestor performs a certification protocol with the TTP:

1. TAS and TTP perform a conventional attestation protocol proving the trustworthiness of the attestor's configuration S_0.
2. TAS creates an attestation key pair (sk_{TAS}, pk_{TAS}) and sends the public part, the TPM's endorsement key ek_{TPM} and the vendors TPM certificate to the TTP.
3. The TTP checks whether the certificate is valid and whether S_0 is trustworthy. Then it creates a certificate $cert(sk_{TTP}, pk_{TAS})$ and seals the certificate under the endorsement key ek and the configuration S_0.

4. The TAS unseals the certificate and checks, whether the certified attestation key matches the locally created key. If successful, the TAS hands out the certificate to the user. Now, the TAS is able to convince challengers that the TAS is trustworthy using sk_{TAS}.

A.2 Attestation

The goal of property-based attestation is to convince the challenger that the consumer platform enforces a given property without leaking information about the current platform configuration.

1. The provider sends an attestation challenge including the demanded property, a list of accepted TTPs and fresh nonce to the TAS.
2. The TAS checks whether it owns a certificate stating that the current system configuration is compatible with the demanded property and that it is signed by an accepted TTP. If it has one, it signs the nonce and sends the signature and the certificate of the TTP to the challenger.

A.3 Sealing

Using the TAS, challengers do not have to analyze the trustworthiness of every platform configuration anymore. Instead, they only have to specify the desired property of the attestating platform:

1. The attestor and the challenger have to agree on a security policy.
2. TAS creates a new asymmetric encryption key pair (ek_{TAS}, dk_{TAS}) and signs the public key with its attestation key.
3. The challenger encrypts the content under the encryption key ek_{TAS}, and sends the result together with a list of accepted TTPs and the desired property P to the TAS.
4. TAS checks whether it owns a certificate that states that the current configuration is compatible to the desired property and whether the certifying TTP is accepted by the provider. If the tests were successful, TAS decrypts the content and hand it out to the application.

B. GROUP SIGNATURE REALIZATION

B.1 Initialization

The user has to ask the TTP for a group signature key $SK_{P_i}^{S_0}$ with respect to a configuration S_0 that is compatible to policy P_i. The TTP creates the requested group signature key, seals it under configuration S_0, and sends it back to the user platform.

B.2 Attestation

To attest that a platform enforces a property P_i, the challenger sends a fresh challenge to the user platform which signs the nonce using the group signature key $SK_{P_i}^{S_0}$. If the challenger receives the signed nonce, it can verify the signature using the corresponding public key PK_{P_i}. Since the attestor can access $SK_{P_i}^{S_0}$ only if the platform is in configuration S_0, and if the challenger trusts the TTP, it can trust that the user platform really enforces P_i.