A Protocol for Property-Based Attestation

Liqun Chen*
HP Laboratories
Filton Road, Stoke Gifford
Bristol, BS34 8QZ, UK

Rainer Landfermann, Hans Löhr,
Markus Rohe, Ahmad-Reza Sadeghi,
and Christian Stüble†
Horst Görtz Institute for IT-Security
Applied Data Security Group
Ruhr-Universität Bochum, Germany

ABSTRACT

The Trusted Computing Group (TCG) has issued several specifications to enhance the architecture of common computing platforms by means of new functionalities, amongst others the (binary) attestation to verify the integrity of a (remote) computing platform/application. However, as pointed out recently, the binary attestation has some shortcomings, in particular when used for applications: First, it reveals information about the configuration of a platform (hardware and software) or application. This can be misused to discriminate certain configurations (e.g., operating systems) and the corresponding vendors, or be exploited to mount attacks. Second, it requires the verifier to know all possible "trusted" configurations of all platforms as well as managing updates and patches that change the configuration. Third, it does not necessarily imply that the platform complies with desired (security) properties. A recent proposal to overcome these problems is to transform the binary attestation into property-based attestation, which requires to only attest whether a platform or an application fulfills the desired (security) requirements without revealing the specific software or/and hardware configuration.

Based on previous works, we propose a concrete efficient property-based attestation protocol within an abstract model for the main functionalities provided by TCG-compliant platforms. We prove the security of this protocol under the strong RSA assumption and the discrete logarithm assumption in the random oracle model. Our scheme allows blind verification and revocation of mappings between properties and configurations.

Categories and Subject Descriptors: D.4.6: Security and Protection

General Terms: Algorithms, Design, Security, Verification

*liqun.chen@hp.com

† {landfermann,hloehr,rohe,sadeghi}@crypto.rub.de

STC'06, November 3, 2006, Alexandria, Virginia, USA.

Keywords: TCG binary attestation, security kernels, property-based attestation, zero-knowledge proof of knowledge

1. INTRODUCTION

Today, distributed applications processing security critical data pose sophisticated functional and security requirements on the underlying computing platforms – in particular, in open network scenarios such as the Internet. Distributed applications involve different parties (companies, end-users, content providers, etc.) with possibly conflicting (security) requirements and interests. To cope with this situation, we need mechanisms which provide and maintain the required security services in the sense of multilateral security. Obviously, the applications and the underlying computing platforms need to provide a certain degree of "trustworthiness" that each of the involved parties requires. In practice, this trustworthiness may be determined by verifying the integrity of the corresponding platform/application where a positive result should imply that the platform/application has not been tampered with, and hence, the critical information processed will not leave the intended trust domains.

Verifying the integrity of a platform or an application locally can be implemented, e.g., by a secure boot process and a trusted Graphical User Interface (tGUI) that ensures a trusted path to the application. However, these solutions are insufficient for remote platform or application integrity verification. Remote integrity verification mechanisms may also enable an external party (a remote machine) to verify whether a platform/application *behaves* according to certain security policies.

In this context, Trusted Computing (TC) technology provides the basis for a new generation of computing platforms with new security-relevant architectures both in hardware and software. A well-known initiative promoting this technology is TCG (Trusted Computing Group), an alliance of a large number of IT enterprises[1]. The stated goal of TCG is to provide mechanisms for improving the security and trustworthiness of computing platforms [21, 22, 31, 30]. TCG has published a number of specifications, in particular for the core components, the *Trusted Platform Module* (TPM) [36, 35] and its library *Trusted Software Stack* (TSS) [15]. The current implementation of the TPM is a tamper-evident hardware chip that provides a limited number of cryptographic functionalities. Currently, many vendors ship their computer platforms with a TPM chip. The

[1] www.trustedcomputinggroup.org

new functions extend the conventional PC architecture by mechanisms to (i) protect cryptographic keys, (ii) generate random numbers in hardware, (iii) cryptographically bind (sensitive) data to certain information, e.g., the system configuration and the identifier of the invoking application (*sealing*), and (iv) authenticate a platform/application *(remote attestation)*.

The TCG solution for platform authentication is sometimes called *binary attestation*, since loosely speaking, it measures all the code executed by using certain metrics (currently a cryptographic hash value over the code binary). The result is stored in special registers in the TPM before executing the code. This procedure is bootstrapped starting with a kind of pre-BIOS that is trusted by default and measures the bootloader, storing the result. This procedure builds the so-called *chain of trust* which can then be extended to the operating system components up to applications.

Binary attestation, however, has several shortcomings, in particular for application attestation, regarding flexibility and security/privacy as pointed out in [27, 29]: First, revealing the system configuration may lead to privacy violations and discrimination against the underlying system since the remote party may exclude them from his/her business model, e.g., configurations related to alternative operating systems such as Linux. Note that similar approaches can be observed today, e.g., many banks provide banking software for only one operating system and some music players expect a specific operating system. It then reveals the complete information about the hardware and software configuration of a platform which makes attacks easier. Second, it requires the verifier to know all possible "trusted" configurations of all platforms.[2] Third, it does not necessarily imply that the platform complies with desired (security) properties.

In contrast to binary attestation, a more general and flexible solution is to use *property-based attestation* to attest properties of the underlying platform or/and application instead of revealing the binary information about them. Loosely speaking, 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 for Multilevel Security or privacy protection, or it has a security kernel providing isolation of applications). Attesting properties means that the attestation should only determine whether a platform (or its configuration) fulfills a desired property, instead of revealing the concrete configuration of its software and hardware components. Attesting properties has the advantage that different platforms with different components may have different configurations while they may all offer the same properties and consequently fulfill the same requirements. In particular, this solution also allows a more flexible way of handling system patches and updates.

In this context it is important one needs to define which properties are useful and reasonable, and how to determine them automatically. The former depends strongly on the underlying use case and its requirements and the latter may be performed in different ways in practice as we briefly consider in Section 5.1.

Our proposal for property-based attestation in this paper is based on the ideas in [29], where the authors informally discuss several solutions which differ in their trust models, efficiency and the functionalities offered by the trusted components. We will consider other related works in Section 2. The core idea presented in [29] is what we call *delegation-based* property attestation. Here, a certification agency certifies the mapping between properties and configurations and publishes these *property certificates*. After this, the agency remains completely offline. Now a platform/application claiming to provide a certain property can download the appropriate certificate(s), and prove to any (correct) verifier that it has a valid certificate, since its configuration matches the one fixed in the property certificate without disclosing any information about the content of the certificate (except the property which is a common input which is public by definition). In particular, we are interested in TCG compliant solutions allowing to use TSS and the existing TC hardware without a need to change the underlying trust model of TCG.[3]

Our contribution. We propose a provably secure protocol for property-based attestation that concretely implements the delegation-based solution sketched above. Our solution is based on the current TCG specification, because today it is the most widely known and available extension of conventional computer systems that provides the measurement and attestation of integrity metrics. Hence, our scheme uses a hybrid approach where a property attestor (e.g., a service of a security kernel) calls on a binary attestor (here the TPM). Our scheme allows the revocation of invalid configurations either from a public list or negotiated between the prover (platform) and a verifier. The deployed cryptographic schemes are based on CL signatures [6] and signature proofs of knowledge similar to those in DAA [4]. The property revocation protocol is based on [7].

2. RELATED WORK

There have been several proposals in the literature for protecting and proving the integrity of computing platforms based on cryptographic techniques and trusted components. Known aspects in this context are secure and authenticated (or trusted) booting: the former means that a system can measure its own integrity and terminates the boot process in case the integrity check fails, whereas the latter aims at proving the platform integrity to a (remote) verifier (for both topics see, e.g., [1], [11]).

In [29], and later in [27], the authors propose an approach called *property attestation* to prevent the deficiencies of the existing binary attestation. The basic idea in [27] is to engage a protocol between verifier and attestor to prove that the attested platform satisfies the verifier's security require-

[2]Other problems related to attestation are updates/patches and backup: the new functionalities allow to seal critical data (e.g., documents, content) to a certain platform configuration. This, however, strongly limits the flexibility when system updates (e.g., patches) change the system configuration. As a consequence, the data is not accessible anymore. Similar situations arise with system backups.

[3]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.

ments. Their solution is based on property certificates that are used by a *verification proxy* to translate binary attestations into property attestations. Moreover, this work briefly discusses two deployment scenarios: The verification proxy as a dedicated machine and the verification proxy on the verified platform. Whereas [27] proposes a high-level protocol for property-based attestation, [29] proposes and discusses several protocols and mechanisms that differ in their trust models, efficiency and the functionalities offered by the trusted components.

The authors of [16] propose *semantic remote attestation* – using language-based trusted virtual machines (VM) to remotely attest high-level program properties. The general idea behind this approach is the use of a trusted virtual machine that checks the security policy of the code that runs within the VM. Since the trusted VM still has to be binary attested, semantic remote attestation is a hybrid solution with code analysis.

In [18], [20], and [19] the authors propose a software architecture based on Linux providing attestation and sealing. The architecture allows to bind short-lifetime data (e.g., application data) to long-lifetime data (e.g., the Linux kernel) and to allow access to the data only if the system is compatible with a security policy certified by a security administrator. Moreover, these papers suggest to use a certification authority that certifies the trustworthiness of certain configurations of long-lifetime data. Thus, the proposed architecture is very similar to a hybrid approach based on property certificates as we use in this paper.

3. TCG MAIN COMPONENTS

The core specification of Trusted Computing Group (TCG) concerns the Trusted Platform Module [36, 35], a component which provides certain cryptographic functions. The assumption is that this party is fully trusted. The current implementation of the TPM is a tamper-evident hardware chip. Other major components of the TCG proposal are a kind of (protected) pre-BIOS (Basic I/O System) called the *Core Root of Trust for Measurement* (CRTM), and a support software called *Trusted Software Stack* (TSS) which performs various functions like communicating with the rest of the platform or with other platforms.

Trusted Platform Module & Platform Configuration. A TPM provides a secure random number generator, non-volatile tamper-resistant storage, key generation algorithms, and cryptographic functions for encryption/decryption, digital signatures (RSA) and a cryptographic hash function (SHA-1).

Moreover, the TPM includes a set of registers called *Platform Configuration Registers* (PCR) which can be used to store hash values.

Integrity Measurement. The so-called *Integrity Measurement* is done during the boot process by computing a cryptographic hash of the initial platform state. For this purpose, the CRTM computes a hash of ("measures") the code and parameters of the BIOS and extends the first PCR register by this result according to $PCR_{i+1} \leftarrow \text{SHA1}(PCR_i|\text{Input})$.

A *chain of trust* is established, if, additionally, both BIOS and bootloader measure the code they are executing as well. Hence, $PCR_0, \ldots, PCR_n$ provide evidence of a certain state of the system immediately after the boot process, which we define as the platform's *configuration specification*, denoted abstractly by $cs := (PCR_0, \ldots, PCR_n)$.[4]

Attestation. The TCG attestation protocol is used to give assurance about the platform configuration cs to a remote party. Here the attesting party, the *attestor*, reports to a remote party, the *verifier*, the configuration of a machine(s) to be attested, e.g., the configuration of the platform or/and of applications. To guarantee integrity and freshness, this value and a fresh nonce N_v must be digitally signed with an asymmetric key called *Attestation Identity Key* (AIK) that is under the sole control of the TPM. A trusted third party called *Privacy Certification Authority* (Privacy-CA) is used to guarantee the pseudonymity of the AIKs. However, this party can always link the transactions a certain platform was involved in. To overcome this problem, version 1.2 of the TCG specification [35] defines another cryptographic protocol called *Direct Anonymous Attestation (DAA)* [4] that, roughly spoken, provides users with an unlimited number of pseudonyms without requiring a Privacy-CA. Note that the anonymity provided by DAA or Privacy-CAs is completely orthogonal to the stated goals of this paper. Nevertheless, we will show in Section 7 how both approaches can be combined into an (unlinkable) property-based attestation function.

TCG (implicit) Assumptions. The functionality mentioned above is provided based on the following assumptions: First, the platform configuration cannot be overwritten after measurements (i.e., after the hash values are computed and securely stored in the TPM). Since the TCG makes statements about the initial state of the platform only, it is crucial that this state cannot be (maliciously) manipulated after startup, otherwise a verifier cannot rely on the information provided by the attestor. However, currently available operating systems (e.g., Windows, Linux) can easily be manipulated by exploiting security bugs or by changing memory which has been swapped to a harddisk. Second, given a valid set of hash values, the challenger is able to determine whether the platform configuration is trustworthy. However, the trusted computing base of today's operating systems is very complex, making it very hard, if not impossible, to determine their trustworthiness. Third, the following secure channels can be established: (i) between hardware components (e.g., between TPM and CPU) since both components are integrated on the same hardware;[5] (ii) between the attestor and the verifier, which can reasonably be achieved by a public key infrastructure (PKI) [35], and (iii) between the attestor and the attested machine, which should be provided by the underlying operating system.

Hence, a secure operating system is required that (i) effectively prevents unauthorized modifications, (ii) is small enough to allow an evaluation of its trustworthiness, (iii) provides secure interprocess communication mechanisms, and

[4]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 [34].

[5]Experience shows that this assumption does not hold for the currently available TPM platforms, since it is possible to observe resp. modify the communication between CPU and TPM.

(iv) last but not least, it should be compatible to the legacy software. The recent development in the field of security kernel design (based on e.g., hypervisors or microkernels) shows that these properties can be efficiently provided (see, e.g., [25, 28, 12] and [33, 32, 2, 3]).

4. DEFICIENCIES OF TCG ATTESTATION

As already mentioned in the introduction, while the attestation (and sealing) mechanisms provided by the TCG allow many interesting applications (see, e.g., [31, 14, 17, 34]), the naive use of the platform configuration (e.g., to bind short-term data to platforms or to determine the trustworthiness of applications) has some drawbacks, like the following: the first problem is the potential to be misused for *discrimination*, e.g., to isolate "alternative" software products (e.g., OpenOffice[6] or WINE[7] and operating systems such as Linux). It is imaginable that global players such as content providers and large operating system vendors collaborate and exclude specific operating systems as well as applications. This barrier to entry effectively undermines competition and prevents the self-regulating mechanisms of an open market. The second problem is *complexity* since the number of different platform configurations grows exponentially with the number of patches, compiler options and software versions. This makes it hard to keep track of the trustworthiness of a given configuration. The third problem is *observability*, since the recipient of the attestation protocol or an observer obtains exact information about the hard- and software configuration of a specific platform. This makes attacks on such platforms much easier since an adversary does not need to perform any platform analysis. A further problem is the *scalability*, since update and patches lead to configuration changes.

5. PROPERTY-BASED ATTESTATION

A more general and flexible solution to the attestation problem is a *property-based attestation (PBA)* approach [29, 27]. It means that attestation should only determine whether a platform (configuration) or an application has the desired property. This avoids revealing the concrete configuration of software and hardware components. For example, it would not matter whether the application was Webbrowser A or B, as long as both have the same properties. In contrast, the binary attestation function provided by TCG-compliant hardware attests the system configuration of a platform that was determined at system startup. For (nearly) all practical applications, the verifier is not really interested in the specific system or application configuration. As we pointed out in Section 4, this even has a disadvantage due to the multitude of possible configurations a verifier has to manage.

Informally, a *property*, in this context, describes an aspect of the behavior of the underlying object (platform / application) with respect to certain requirements, e.g., a security-related requirement. In general, properties for different abstraction levels are imaginable. For instance, a platform property may state that a platform is *privacy-preserving*, i.e., it has built-in measures conform to the privacy laws, or that the platform provides *isolation*, i.e., strictly separates processes from each other, or it provides *Multi-Level Security* (MLS) and so forth.

[6]www.openoffice.org
[7]www.winehq.org

The question whether there is a correct or useful property set depends strongly on the underlying use case and its requirements. Attesting properties has the advantage that different platforms with different components may have different configurations while they all may offer the same properties and consequently fulfill the same requirements.[8]

5.1 Determining Properties

Before the attestor can make statements about a machine, the appropriate properties have to be determined. We may group possible mechanisms for determining the properties of a machine into the following categories such as *code control*, *code analysis* or *delegation*. Code control requires a trusted attestor enforcing a machine to behave as expected (e.g., the attestor compares the I/O behavior of the machine with that defined by the desired property p). An example would be to use SELinux as a reference monitor and to attest both SELinux and the enforced security policy, as described in [19]. Code analysis requires the property attestor to directly analyze the code of the targeted machine to derive properties, or alternatively, it verifies whether the machine provides the claimed properties. Examples in this context are proof-carrying code [23] and semantic code analysis [16]. The delegation approach requires the property attestor to prove that another trusted third party has certified the desired properties. Obviously, this third party has to be trusted by both the platform to be attested and the verifier. A practical example in this context are property certificates issued by a certificate issuer: The property attestor proves that a property certificate exists and was issued by a third party which is trusted by the verifier.

In this paper, we follow the delegation-based approach with an offline trusted third party. Section 5.2 details this approach and the underlying abstract model of the system.

5.2 Delegation-Based Solution

In this section, we explain the general idea of the delegation-based property-based attestation. Figure 1 illustrates

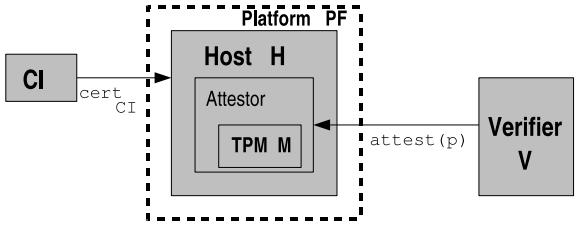

Figure 1: Abstract model of the attestation scenario with certificate issuer $\mathcal{CI}$, platform $\mathcal{PF}$, host $\mathcal{H}$, TPM $\mathcal{M}$ and verifier $\mathcal{V}$

the abstract model with certificate issuer, platform, host, attestor, TPM and verifier. In the following, we introduce the involved roles.

Roles. A *platform*, denoted by $\mathcal{PF}$, represents our main IT system, i.e., it consists of all (software and hardware) components of a system. The *Trusted Platform Module* (TPM) is denoted by $\mathcal{M}$, and is one of the main components of a

[8]One may consider the desired properties of an application as a certain input/output behavior.

platform $\mathcal{PF}$. The TPM has a predefined set of computational and cryptographic capabilities (see Section 3) and is trusted by all parties. A *host* $\mathcal{H}$ is the other main component of $\mathcal{PF}$, in which a TPM $\mathcal{M}$ is embedded. The host includes the software running on the platform $\mathcal{PF}$. The TPM can only communicate with other parties (external to the platform) via the host. A *verifier* is denoted by $\mathcal{V}$ and is a party that wants to verify the attestation result of some platform. The *certificate issuer*, denoted by $\mathcal{CI}$, is the party that certifies mappings between properties and configurations attesting that a given platform configuration *cs* fulfills a desired property p by means of a property certificate $\sigma_{\mathcal{CI}}$ (see Figure 1).

Note that for security protocols, such as PBA or DAA, a trusted component (trusted by the platform or platform owner) is needed within the host that can establish secure channels to the attestor.[9] More precisely, this component must belong to the Trusted Computing Base (TCB). Otherwise, the host can easily disclose to the verifier the configuration of the corresponding platform or/and application in the context of PBA (or the TPM identity in the context of DAA).

The delegation-based principle is well-suited to the TCG trust model and the related infrastructure that already requires trust in third parties (e.g., Privacy-CA, certificate issuer in the context of DAA, or Migration Authority for migratable keys [36, 35]). Our approach is a *hybrid attestation*, which means a two-level chain of attestations, where the first attestation is based on binary configurations (by the TPM) and the second one based on properties (by the corresponding PBA service).

For a general property-based attestation, we assume in our model that applications are attested by the operating system. We stress that in this way, we only need to establish a trusted attestation service on top of a binary attestor (here TPM) still being conform to TCG. We do not elaborate on this service at this stage due to space restrictions and only consider the cryptographic proof protocols for proving the possession of a valid property-certificate conform to the platform's configuration.

Note that $\mathcal{CI}$ confirms the correctness of the correspondence between the platform configuration and certain properties according to defined criteria. However, following common practice, such organizations are only liable for intentional misbehavior and not for undetected weaknesses (compare with safety and security tests or common criteria). Parties like $\mathcal{CI}$ are fully trusted, i.e., by the attestor and the verifier, since both have to assume that $\mathcal{CI}$ certifies only configurations that really have the attested property.

To prevent a flood of desired properties, the involved parties can, e.g., define earmarked property profiles together. For instance, for end-users one could define a privacy-protecting Common Criteria [10] 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 the consumer's privacy, e.g., by certifying that it is compatible to privacy laws.

[9]This trusted component could be a service of a trustworthy operating system.

6. BUILDING BLOCKS

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

Notation. Let $\{0,1\}^\ell$ denote the set of all binary strings of length ℓ which we identify with the integers out of $[0; 2^\ell[$. A protocol $\texttt{Prot}()$ consists of two or more parties, each performing local computations and exchanging messages with other participants. These parties are modeled as polynomial interactive algorithms. A protocol $\texttt{Prot}(\mathcal{P}_1, \mathcal{P}_2)$ with the two participants $\mathcal{P}_1$ and $\mathcal{P}_2$ is characterized by the following input and output parameters: The common input c is given to both involved parties $\mathcal{P}_1$ and $\mathcal{P}_2$, while the private input $in_{\mathcal{P}_1}$, $in_{\mathcal{P}_2}$ is only available to $\mathcal{P}_1$ or $\mathcal{P}_2$, respectively. When both parties have finished the execution of the protocol, each of them obtains its individual output $out_{\mathcal{P}_1}$, $out_{\mathcal{P}_2}$. Outputs may include an indicator $ind \in \{\top, \bot\}$ indicating that the corresponding party accepts/rejects. An execution (or run) of a protocol is denoted by

$$(\mathcal{P}_1 : out_{\mathcal{P}_1};\ \mathcal{P}_2 : out_{\mathcal{P}_2}) \leftarrow \texttt{Prot}(\mathcal{P}_1 : in_{\mathcal{P}_1};\ \mathcal{P}_2 : in_{\mathcal{P}_2};\ c).$$

We mark a malicious party, i.e., a party that is controlled by an adversary, with a $*$, like $\mathcal{P}^*$. Note that if the protocol can be executed by one single party (i.e., non-interactively), we omit the party's name.

The proposed PBA scheme makes use of the following cryptographic primitives:

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

TPM Signatures. The TPM can create a TPM signature $\sigma_{\mathcal{M}}$. The existing TCG technology provides two types of TPM signatures. The first is DAA signatures [4]. With a DAA signature, a verifier is convinced that a TPM has signed a given message, which is either an Attestation Identity Key (AIK) or an arbitrary data string, but the verifier cannot learn the identity of the TPM. The second is ordinary RSA-type signatures. A TPM RSA signature is signed under an AIK, which could either be certified by a Privacy-CA or be introduced by the TPM itself using a DAA signature. For simplicity, we do not distinguish these two cases, and denote the private signing key used to create σ_M by $sk_{\mathcal{M}}$ and denote the corresponding public verification key used to verify $\sigma_{\mathcal{M}}$ by $vk_{\mathcal{M}}$. We denote the TPM signing algorithm on a message m by $\sigma_{\mathcal{M}} \leftarrow \texttt{SignM}(sk_{\mathcal{M}}, m)$.

Commitment Scheme. We apply the commitment scheme by Pedersen [24]: Let $sk^m_{\texttt{com}}$ be the secret commitment key. A commitment on a message m is computed as $C_m := g^m h^{sk^m_{\texttt{com}}} \bmod P$. P is a large prime, h is a generator of a cyclic subgroup $G_Q \subseteq \mathbb{Z}^*_P$ of prime order Q and $Q|P-1$. g is chosen randomly from $\langle h \rangle$; furthermore, $\log_h(g)$ is unknown to the committing party. Both the message m and $sk^m_{\texttt{com}}$ are taken from $\mathbb{Z}_Q$. The Pedersen commitment scheme as described

above is perfectly hiding and computationally binding under the discrete logarithm assumption.

CL Signatures. We apply a variant of the Camenisch and Lysyanskaya (CL) signature scheme [6], as already used in [4], for signing a tuple of messages $X := (x_1, \ldots, x_m)$, where $x_i \in \{0,1\}^{\ell_x}$ $(i = 1, \ldots, m)$ and ℓ_x denotes the maximum binary length for each x_i. In our specification, the CL signing algorithm is denoted as $(A, e, v) \leftarrow \texttt{Sign}(sk; X)$, where sk is an RSA-type private key created from strong primes. The signature (A, e, v) satisfies $Z \equiv A^e \cdot R_0^{x_0} \cdot \ldots \cdot R_m^{x_m} \cdot S^v \mod n$, where the value $e \in [2^{\ell_e - 1}, 2^{\ell_e - 1} + 2^{\ell'_e - 1}]$ is a random prime, the value v of length ℓ_v is a random integer, and the tuple $(R_0, \ldots, R_m, S, Z, n)$ is the corresponding public key vk. The CL signature verification algorithm is denoted as $ind \leftarrow \texttt{Verify}(vk; X, A, e, v)$.

In [5], it is remarked that the CL signature has the ability to be *randomized*. This means that the signature (A, e, v) can be masked to $(\hat{A} := AS^w, e, \hat{v} := v - we)$ with an arbitrary value w. From the verifier's point of view, $(\hat{A}, e, \hat{v})$ and (A, e, v) are both valid signatures on X.

Zero-knowledge Proofs of Knowledge. Zero-knowledge proofs of knowledge are interactive protocols carried out between two parties: a prover and a verifier. During such a protocol run, a verifier is convinced with overwhelming probability that the prover is aware of some secret and that a certain predicate related to this secret is true. However, the verifier does not learn anything beyond this assertion.

Several protocols in this paper will contain some proofs of knowledge of relations among discrete logarithms under exponential one-way homomorphisms. To describe the semantics of these proofs we apply the notation suggested by Camenisch and Stadler [8]. For example,

$$PK\{(\alpha, \beta) : g^\alpha h^\beta \wedge \alpha \in [a, b]\}$$

denotes a zero-knowledge proof of knowledge that a prover is aware of some secret values α and β such that $y = g^\alpha h^\beta$ holds and, moreover, that α is contained in the interval $[a, b]$. g, h, y are elements of some group G with $\langle g \rangle = \langle h \rangle = G$ provided as common input to both parties; this holds for the integers a and b as well.

Depending on the cryptographic assumption the one-way property of a given homomorphism is based on[10], the *soundness* of the corresponding zero-knowledge proof is valid under the same assumption[11]. Furthermore, all occurring proofs of knowledge feature the *statistical zero-knowledge* property. In the case where the verifier chooses the challenge uniformly at random, we obtain a *honest verifier zero-knowledge* (HVZK) proof of knowledge protocol.

According to the Fiat-Shamir Heuristic [13], such a proof of knowledge protocol can be transformed into a non-interactive signature for a message m which will be denoted as $SPK\{(\alpha) \,|\, y = g^\alpha\}(m)$.

[10] here either strong RSA assumption for CL signatures or the discrete logarithm assumption for Pedersen's commitment scheme

[11] provided that the verifier's challenge is chosen smaller than the smallest factor of the underlying group's order (either $QR(n)$ or G_Q)

7. THE PROPOSED PROTOCOL FOR PBA

In this solution, we describe a concrete property-based attestation protocol, which consists of property certificates, a PBA signing algorithm, a verification algorithm and a revocation check process. This protocol holds the security properties of *unforgeability* and *unlinkability*. Informally, unforgeability means that a PBA signature can only be produced with the involvement of a valid TPM to the actual platform configuration; unlinkability means that from the PBA signature and its verification protocol, a verifier is not able to deduce the specific configuration of the platform.

The basic idea is as follows: the host $\mathcal{H}$ proves that there is a valid link between the conventional binary attestation signature $\sigma_{\mathcal{M}}$, generated by the trusted component (here the TPM), and the certificate (represented by the signature $\sigma_{\mathcal{CI}}$) of a certificate issuer $\mathcal{CI}$ attesting that the configuration specification cs_i provides the property specification denoted by ps. Here, the prover obtains the corresponding certificate $\sigma_{\mathcal{CI}}$ as secret input, and the verifier takes the public key $vk_{\mathcal{CI}}$ of the certificate issuer and the property specification ps as individual input. The prover proves directly that its configuration complies with the one in the certificate without showing the certificate.

Note that the revocation process in this protocol does not involve a trusted party. A prover can convince a verifier that its configuration is not among a given set of revoked configurations. It is not necessary for a trusted third party to provide the set of revoked configurations, which could be negotiated directly between the prover and verifier.

7.1 Security Parameters

In this section, we enumerate the security parameters $\ell_x(y)$ used in the PBA protocol specified below with their required bitlength y. ℓ_{cs} (160) indicates the size of a configuration value cs, while ℓ_{ps} (160) determines the binary length of a certain property ps. $\ell_\varnothing$ (80) denotes the security parameter controlling the statistical zero-knowledge property and $\ell_{\mathcal{H}}$ (160) is the output length of the hash function used for the Fiat-Shamir heuristic. For Pedersen's commitment scheme, the size of the modulus P is set to ℓ_P (1632) and the size of the order Q of the sub group of $\mathbb{Z}_P^*$ to ℓ_Q. The parameters ℓ_P and ℓ_Q should be chosen such that the discrete logarithm problem in the subgroup of $\mathbb{Z}_P^*$ of order Q, with P and Q being primes such that $2^{\ell_Q} > Q > 2^{\ell_Q - 1}$ and $2^{\ell_P} > P > 2^{\ell_P - 1}$, has acceptable computational difficulty. Furthermore, ℓ_n (2048) indicates the size of an RSA modulus, while ℓ_e (368) and ℓ'_e (120) are parameters occurring in the blinded CL signature scheme. Finally, ℓ_v (2536) is the size of v, a random value which is part of the certificate.

Moreover, we require the following constraints among the security parameters:

$$\begin{aligned} &\ell_Q > \ell_{cs} + \ell_{\mathcal{H}} + \ell_\varnothing + 2, \\ &\ell_e \geq \max\{\ell_{cs}, \ell_{ps}\} + 2, \quad \text{and} \\ &\ell_v \geq \ell_n + \max\{\ell_{cs}, \ell_{ps}\} + \ell_\varnothing. \end{aligned}$$

7.2 Property-Configuration Certificates

An acceptable configuration attestation is certified by a certificate issuer $\mathcal{CI}$. The procedures of key generation, certificate issuing and verification are described in Figure 2.

We denote the corresponding protocols with

$$(sk_{\mathcal{CI}}, vk_{\mathcal{CI}}) \leftarrow \texttt{KeyGen}(1^{\ell_n}),$$
$$\sigma_{\mathcal{CI}} \leftarrow \texttt{IssueCertCI}(sk_{\mathcal{CI}}, (cs_i, ps)), \quad \text{and}$$
$$ind \leftarrow \texttt{VerifyCertCI}(vk_{\mathcal{CI}}, (cs_i, ps), \sigma_{\mathcal{CI}}).$$

7.3 Signing Algorithm

The signature procedure is a protocol among the TPM $\mathcal{M}$ and the host $\mathcal{H}$ and is presented in Figure 3. As a result of the protocol, the host will have created a masked signature $\sigma_{\texttt{PBA}}$, which is based on a TPM signature $\sigma_{\mathcal{M}}$ on the message C_{cs_i}, where C_{cs_i} is the commitment to configuration specification cs_i. From the masked signature, the verifier will be convinced that the platform has a valid configuration associated with a given property ps. The protocol is denoted by

$$(\mathcal{M}\colon \sigma_{\mathcal{M}};\ \mathcal{H}\colon \sigma_{\texttt{PBA}}) \leftarrow$$
$$\texttt{PBASign}(\mathcal{M}\colon sk_{\mathcal{M}};\ \mathcal{H}\colon \sigma_{\mathcal{CI}};\ vk_{\mathcal{CI}}, par_{\texttt{com}}, cs_i, ps, N_v),$$

where the commitment parameters are $par_{\texttt{com}} := (g, h, P, Q)$ and the public verification key of certificate issuer $\mathcal{CI}$ is $vk_{\mathcal{CI}} = (n, R_0, R_1, S, Z)$.

7.4 Verification Algorithm

The verification protocol (see Figure 4) checks if a given signature $\sigma_{\texttt{PBA}}$ is correct, i.e., whether the input to the signing protocol was a configuration with a valid property certificate and a correct TPM signature. The protocol is denoted by $ind \leftarrow \texttt{PBAVerify}(vk_{\texttt{PBA}}, \sigma_{\texttt{PBA}}, N_v)$, where $vk_{\texttt{PBA}} := (vk_{\mathcal{CI}}, vk_{\mathcal{M}}, par_{\texttt{com}}, ps)$ is the verification key corresponding to the signature $\sigma_{\texttt{PBA}}$.

7.5 Revocation Check of a Certificate

If for any reason, e.g., due to system security updates, a set of configuration specifications becomes invalid, the corresponding list will be published (usually by $\mathcal{CI}$) so that possible verifiers can access it.

Suppose that $CS_{\text{revoked}} = \{cs_j\}_{j=1,\ldots,\tau}$ is the set of invalid configuration specifications, either from a public list or negotiated between prover and verifier. Then the revocation check protocol in Figure 5, performed by a host $\mathcal{H}$ and a verifier $\mathcal{V}$, checks whether the configuration which was used for signing some property is contained in the set of revoked configurations CS_{revoked}. This protocol is denoted by

$$(\mathcal{V}\colon ind;\ \mathcal{H}\colon \sigma_R) \leftarrow$$
$$\texttt{PBARevoke}(\mathcal{V}\colon -;\ \mathcal{H}\colon cs_i, r;\ par_{\texttt{com}}, f, C_{cs_i}, CS_{\text{revoked}}),$$

where cs_i is the host configuration, $r = sk^i_{com}$ is the key for opening the commitment, $f \neq g \neq h$ is a generator of G_Q, and C_{cs_i} is the host's commitment to the configuration cs_i.

Technically, this proof is derived from a zero-knowledge protocol to prove the inequality of two discrete logarithms presented in [7]. The host shows that the configuration cs_i, to which it committed during $\texttt{PBASign}()$, is not equal to any of the revoked configurations cs_j. To achieve this, he proves the knowledge of both exponents in the commitment (i.e., cs_i and r) and that $cs_i \neq cs_j \quad (\forall j = 1, \ldots, \tau)$. To prove these inequalities, the technique introduced in [7] by Camenisch and Shoup is used. If $cs_i = cs_j$ for some j, the verifier would notice this in step 5e, because in this case he would get $D_j = 1$.

7.6 Rogue TPMs

If a TPM is broken, there must be a possibility to recognize this. In our scheme, the TPM signs the configuration by using a signing key, usually an AIK. If this AIK has been signed using a DAA signature, then the tagging will be covered by the DAA rogue tagging, and a potential verifier can check this. If other mechanisms such as a Privacy-CA are used to certify the AIK, then this AIK will be put on revocation lists. In any case, the problem of rogue tagging corrupted TPMs is out of scope of the PBA protocol.

7.7 Security Analysis

We analyzed the security of the PBA protocol in a formal model based on the simulatability paradigm, as proposed in [9] and [26]. The ideal-world specification fulfills the requirements *unforgeability* and *unlinkability* – and, according to the proof, the real-world protocol implements this specification. The security proof of DAA [4] uses a similar approach. We show that the security of the PBA protocol relies on CL signatures (cf. [6]) and Pedersen commitments (cf. [24]), which in turn are based on the strong RSA assumption and the discrete logarithm assumption in the corresponding algebraic structures (see Section 6). We summarize this result in the following theorem.

Theorem 1 *The above protocol implements a secure property-based attestation system under the discrete logarithm assumption and the strong RSA assumption in the random oracle model.*

A formal definition of the PBA security model and the proof of Theorem 1 is given in the extended version of this paper[12]. The basic idea of the proof is to specify an ideal world, where a trusted party $\mathcal{T}$ ensures the security properties. The protocol is said to be secure if for every (polynomially bounded) adversary in the real world, there exists an adversary in the ideal world, such that both settings cannot be distinguished (except with negligible probability). This is achieved by the introduction of a simulator into the ideal world that has black-box access to the real-world adversary and acts as ideal-world adversary. Such a simulator is constructed independently from the actual adversary and hence works for all possible adversaries. Then the proof is concluded by showing that both worlds are indistinguishable.

8. CONCLUSION AND OUTLOOK

In this paper, we proposed the first cryptographic instantiation of a protocol for property-based attestation based on the TCG trust model. Our approach is based on the delegation principle and uses an offline trusted third party as issuer for property-configuration certificates. Moreover, a flexible dealing with the revocation of invalid platform configurations was proposed, where revocation is handled by a two-party protocol between host and verifier without the involvement of a trusted third party. Finally, we provided a proof of security under well-established assumptions in a formal model using the simulatability paradigm.

A further research topic is the question, how more complex properties can be derived automatically. In practice, it is difficult to be able to determine or compare properties enforced by a platform configuration. In the future, im-

[12] `http://www.prosecco.rub.de/publication.html`

1. **Key generation:** On input 1^{ℓ_n}, create a special RSA modulus $n = pq$ of length ℓ_n where p and q are strong primes. Choose, uniformly at random, R_0, R_1, S, $Z \in QR_n$. Output the public verification key $vk_{\mathcal{CI}} = (n, R_0, R_1, S, Z)$ and the secret signing key $sk_{\mathcal{CI}} = p$. We denote this algorithm with

$$(vk_{\mathcal{CI}}, sk_{\mathcal{CI}}) \leftarrow \texttt{KeyGen}(1^{\ell_n}).$$

2. **Signing algorithm:** Given a property specification $ps \in \{0,1\}^{\ell_{ps}}$ and a set of the corresponding configuration specifications $cs_i \in \{0,1\}^{\ell_{cs}}$ with $i = 1, ..., t$. The property certificate on each configuration specification is issued as follows. On input (cs_i, ps) with $ps \in \{0,1\}^{\ell_{ps}}$ and $cs_i \in \{0,1\}^{\ell_{cs}}$, choose a random prime number e_i of length $\ell_e \geq \max(\ell_{ps}, \ell_{cs})+2$, and a random number v_i of length $\ell_v = \ell_n + \max(\ell_{ps}, \ell_{cs}) + \ell_{\varnothing}$. Compute the value A_i such that $Z \equiv A_i^{e_i} \cdot R_0^{cs_i} \cdot R_1^{ps} \cdot S^{v_i} \bmod n$. The signature on the message (cs_i, ps) is the tuple $\sigma_{\mathcal{CI}} := (A_i, e_i, v_i)$. We denote this algorithm with

$$\sigma_{\mathcal{CI}} \leftarrow \texttt{IssueCertCI}(sk_{\mathcal{CI}}, (cs_i, ps)).$$

3. **Verification algorithm:** Let $i \in \{1, ..., t\}$. To verify that the tuple (A_i, e_i, v_i) is a signature on message (cs_i, ps), check that $Z \equiv A_i^{e_i} \cdot R_0^{cs_i} \cdot R_1^{ps} \cdot S^{v_i} \bmod n$, and check that $2^{\ell_e} > e_i > 2^{\ell_e - 1}$. We denote this algorithm with

$$ind \leftarrow \texttt{VerifyCertCI}(vk_{\mathcal{CI}}, (cs_i, ps), \sigma_{\mathcal{CI}}).$$

Figure 2: Issuing a Certificate of a Property and Configuration Map.

Players: TPM $\mathcal{M}$ and the corresponding Host $\mathcal{H}$
TPM's input: $sk_{\mathcal{M}}$; **Host's input:** $\sigma_{\mathcal{CI}} := (A_i, e_i, v_i)$.
Common input: $vk_{\mathcal{CI}} = (n, R_0, R_1, S, Z)$, $par_{\texttt{com}} := (g, h, P, Q)$, cs_i, ps, N_v (nonce provided by the verifier).

1. The TPM performs as follows
   (a) Choose a random $N_t \in_R \{0,1\}^{\ell_{\varnothing}}$.
   (b) Choose a random $r \in_R \{0,1\}^{\ell_Q}$ and compute the commitment $C_{cs_i} := g^{cs_i} h^r \bmod P$.
   (c) Generate a TPM signature $\sigma_{\mathcal{M}} := \texttt{SignM}(sk_{\mathcal{M}}, par_{\texttt{com}} \| C_{cs_i} \| N_v \| N_t)$.
   (d) Send to host $\sigma_{\mathcal{M}}$ with the values C_{cs_i}, r and N_t.
2. The host performs the following steps to finish the proof:
   (a) Choose at random $w \in \{0,1\}^{\ell_n}$.
   (b) Compute $\hat{A} = A_i S^w \bmod n$ and $\hat{v} = v_i - we_i$.
   (c) Create a masked signature $\hat{\sigma}_{\mathcal{CI}} = (\hat{A}, e_i, \hat{v})$.
3. The host computes the signature of knowledge protocol

$$SPK\{(cs_i, e_i, \hat{v}, r) : Z/R_1^{ps} \equiv \pm \hat{A}^{e_i} R_0^{cs_i} S^{\hat{v}} (\bmod\ n) \wedge C_{cs_i} = g^{cs_i} h^r (\bmod\ P)$$
$$cs_i \in \{0,1\}^{\ell_{cs}+\ell_{\varnothing}+\ell_{\mathcal{H}}+2} \wedge (e_i - 2^{\ell_e}) \in \{0,1\}^{\ell'_e+\ell_{\varnothing}+\ell_{\mathcal{H}}+1}\}(N_v, N_t)$$

   in the following steps:
   (a) Computes $Z' = Z/R_1^{ps} \bmod n$.
   (b) Pick random integers

$$r_v \in_R \{0,1\}^{\ell_e+\ell_n+2\ell_{\varnothing}+\ell_{\mathcal{H}}+1},\ r_e \in_R \{0,1\}^{\ell'_e+\ell_{\varnothing}+\ell_{\mathcal{H}}},\ r_{cs} \in_R \{0,1\}^{\ell_{cs}+\ell_{\varnothing}+\ell_{\mathcal{H}}},\ r_r \in_R \{0,1\}^{\ell_Q+\ell_{\varnothing}+\ell_{\mathcal{H}}}.$$

   (c) Compute $\tilde{Z}' := \hat{A}^{r_e} R_0^{r_{cs}} S^{r_v} \bmod n$ and $\tilde{C}_i := g^{r_{cs}} h^{r_r} \bmod P$.
   (d) Compute $c := \texttt{Hash}(vk_{\mathcal{CI}} \| par \| ps \| C_{cs_i} \| \tilde{Z}' \| \tilde{C}_i \| N_v \| N_t)$.
   (e) Compute $s_v := r_v + c \cdot \hat{v}$, $s_{cs} := r_{cs} + c \cdot cs_i$, $s_e := r_e + c \cdot (e_i - 2^{\ell_e - 1})$ and $s_r := r_r + c \cdot r$ over the integers.
   (f) The PBA signature will be $\sigma_{\texttt{PBA}} := (\hat{A}, \sigma_{\mathcal{M}}, N_t, C_{cs_i}, c, s_v, s_{cs}, s_e, s_r)$.

Figure 3: The PBA Signing Algorithm.

Players: Verifier $\mathcal{V}$;
Verifier's input: $vk_{\mathcal{CI}}, vk_{\mathcal{M}}, par_{\mathtt{com}}, ps, \sigma_{\mathtt{PBA}}, N_v$. The verifier verifies the signature by performing as follows:

1. Verify $\sigma_{\mathcal{M}}$ w.r.t. $vk_{\mathcal{M}}$ on the message $(par_{\mathtt{com}} \| C_{cs_i} \| N_v \| N_t)$. If positive go to the next step.
2. Compute $\hat{Z}' := Z'^{-c} \hat{A}^{s_e + c2^{\ell_e - 1}} R_0^{s_{cs}} S^{s_v} \bmod n$ and $\hat{C}_i := C_{cs_i}^{-c} g^{s_{cs}} h^{s_r} \bmod P$.
3. Verify that $c \stackrel{?}{=} \mathtt{Hash}(vk_{\mathcal{CI}} \| par \| ps \| C_{cs_i} \| \hat{Z}' \| \hat{C}_i \| N_v \| N_t)$, $s_{cs} \stackrel{?}{\in} \{0,1\}^{\ell_{cs} + \ell_\varnothing + \ell_{\mathcal{H}} + 1}$ and $s_e \stackrel{?}{\in} \{0,1\}^{\ell'_e + \ell_\varnothing + \ell_{\mathcal{H}} + 1}$.

Figure 4: The Verification Protocol.

Players: Host $\mathcal{H}$ and Verifier $\mathcal{V}$.
Common input: $(par_{\mathtt{com}}, f, C_{cs_i}, CS_{\text{revoked}})$, where $par_{\mathtt{com}} := (g, h, P, Q)$ and $CS_{\text{revoked}} = \{cs_j\}_{j=1,...,\tau}$;
Host's input: (cs_i, r).

1. The host and verifier compute $G_j = C_{cs_i} / g^{cs_j} \pmod P$ $(j = 1, ..., \tau)$.
2. The host computes $F = f^r \pmod P$ and sends F to the verifier.
3. The host randomly picks $\beta \in \{0,1\}^{\ell_Q}$, computes $D_j := h^{r\beta} G_j^{-\beta} \pmod P (j = 1, ..., \tau)$ and sends D_j to verifier.
4. The host computes the signature of knowledge by performing as follows:

$$SPK\{(cs_i, r, \alpha, \beta) : C_{cs_i} = g^{cs_i} h^r (\bmod P) \ \wedge F = f^r \pmod P \ \wedge \ 1 = f^\alpha F^{-\beta} \ \wedge \ D_j = h^\alpha G_j^{-\beta} \text{ for } j = 1, ..., \tau\}$$

   (a) Pick random integers

$$r_{cs} \in_R \{0,1\}^{\ell_{cs} + \ell_\varnothing + \ell_{\mathcal{H}}}, \ r_r \in_R \{0,1\}^{\ell_Q + \ell_\varnothing + \ell_{\mathcal{H}}}, \ r_\alpha \in_R \{0,1\}^{\ell_Q + \ell_\varnothing + \ell_{\mathcal{H}}}, \ r_\beta \in_R \{0,1\}^{\ell_Q + \ell_\varnothing + \ell_{\mathcal{H}}}.$$

   (b) Compute $\tilde{C}_i := g^{r_{cs}} h^{r_r} \bmod P$, $\tilde{F} := f^{r_r} \bmod P$, $\tilde{D}_0 := f^{r_\alpha} F^{-r_\beta} \bmod P$ and $\tilde{D}_j := h^{r_\alpha} G_j^{-r_\beta}$ for $j = 1, ..., \tau$.
   (c) Compute $c := \mathtt{Hash}(g \| h \| C_{cs_i} \| F \| \tilde{C}_i \| \tilde{F} \| \tilde{D}_0 \| \tilde{D}_1 \| ... \| \tilde{D}_\tau)$.
   (d) Compute $s_{cs} := r_{cs} + c \cdot cs_i$, $s_r := r_r + c \cdot r$, $s_\alpha := r_\alpha + c \cdot \alpha$ and $s_\beta := r_\beta + c \cdot \beta$. (Note that $\alpha := r \cdot \beta \bmod Q$.)
   (e) Send the signature $\sigma_R := (c, s_{cs}, s_r, s_\alpha, s_\beta)$ to the Verifier.

5. The verifier verifies the signature as follows:

   (a) Verify that $s_{cs} \stackrel{?}{\in} \{0,1\}^{\ell_{cs} + \ell_\varnothing + \ell_{\mathcal{H}} + 1}$.
   (b) Compute $\hat{C}_i := g^{s_{cs}} h^{s_r} / C_{cs_i}^c \bmod P$, $\hat{F} := f^{s_r} / F^c \bmod P$, $\hat{D}_0 := f^{s_\alpha} F^{-s_\beta} \bmod P$, and $\hat{D}_j := h^{s_\alpha} G_j^{-s_\beta} / D_j^c \bmod P \quad (\forall j = 1, ..., \tau)$.
   (c) Compute $\hat{c} := \mathtt{Hash}(g \| h \| C_{cs_i} \| F \| \hat{C}_i \| \hat{F} \| \hat{D}_0 \| \hat{D}_1 \| ... \| \hat{D}_\tau)$.
   (d) If $\hat{c} = c$, accept the proof, otherwise reject the proof.
   (e) If $D_j \neq 1$ holds for $j = 1, ..., \tau$, C_{cs_i} was not committed to any of G_j $(j = 1, ..., \tau)$, the verifier accepts C_{cs_i}.

Figure 5: The Revocation Check Protocol.

proved software-engineering methods based on formal methods, proof-carrying code and semantic code analysis may give the chance to formally or semi-formally derive properties from code directly and thus to prevent the need for a trusted third party.

9. REFERENCES

[1] W. A. Arbaugh, D. J. Farber, and J. M. Smith. A secure and reliable bootstrap architecture. In *Proceedings of the IEEE Symposium on Research in Security and Privacy*, pages 65–71, Oakland, CA, May 1997, IEEE Computer Society Press.

[2] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield. Xen and the art of virtualization. In *SOSP '03: Proceedings of the 19th ACM symposium on Operating systems principles*, pages 164–177, New York, NY, USA, 2003. ACM Press.

[3] P. Barham, B. Dragovich, K. Fraser, S. Hand, A. Ho, and I. Pratt. Safe hardware access with the Xen virtual machine monitor. In *1st Workshop on Operating System and Architectural Support for On-Demand IT Infrastructure*, 2004.

[4] E. Brickell, J. Camenisch, and L. Chen. Direct anonymous attestation. In *Proceedings of the 11th ACM Conference on Computer and Communications Security*, Washington, DC, USA, Oct. 2004. ACM Press.

[5] J. Camenisch and J. Groth. Group signatures: Better efficiency and new theoretical aspects. In C. Blundo and S. Cimato, editors, *SCN*, volume 3352 of *LNCS*, pages 120–133. Springer, 2004.

[6] J. Camenisch and A. Lysyanskaya. A signature scheme with efficient protocols. In *Third Conference on Security in Communication Networks - SCN '02*, volume 2576 of *LNCS*, pages 268–289. Springer-Verlag, Berlin Germany, 2002.

[7] J. Camenisch and V. Shoup. Practical verifiable encryption and decryption of discrete logarithms. In D. Boneh, editor, *CRYPTO*, volume 2729 of *Lecture Notes in Computer Science*, pages 126–144. Springer, 2003.

[8] J. Camenisch and M. Stadler. Proof systems for general statements about discrete logarithms. Technical Report TR 260, Department of Computer Science, ETH Zürich, Mar. 1997.

[9] R. Canetti. Security and composition of multiparty cryptographic protocols. *Journal of Cryptology*, 13(1):143–202, Winter 2000.

[10] 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`.

[11] J. Dyer, M. Lindemann, R. Perez, R. Sailer, L. van Doorn, S. W. Smith, and S. Weingart. Building the IBM 4758 Secure Coprocessor. *IEEE Computer*, 34(10):57–66, 2001.

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

[13] A. Fiat and A. Shamir. How to prove yourself: Practical solutions to identification and signature problems. In A. M. Odlyzko, editor, *Advances in Cryptology – CRYPTO '86*, volume 263 of *LNCS*, pages 186–194, Santa Barbara, CA, USA, 1987. International Association for Cryptologic Research, Springer-Verlag, Berlin Germany.

[14] 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)*, pages 193–206, 2003.

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

[16] V. Haldar, D. Chandra, and M. Franz. Semantic remote attestation: A virtual machine directed approach to trusted computing. In *USENIX Virtual Machine Research and Technology Symposium*, May 2004. also Technical Report No. 03-20, School of Information and Computer Science, University of California, Irvine; October 2003.

[17] 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)*, pages 178–192, 2003.

[18] R. MacDonald, S. Smith, J. Marchesini, and O. Wild. Bear: An open-source virtual secure coprocessor based on TCPA. Technical Report TR2003-471, Department of Computer Science, Dartmouth College, 2003.

[19] J. Marchesini, S. Smith, O. Wild, A. Barsamian, and J. Stabiner. Open-source applications of TCPA hardware. In *20th Annual Computer Security Applications Conference*. ACM, Dec. 2004.

[20] J. Marchesini, S. W. Smith, O. Wild, and R. MacDonald. Experimenting with TCPA/TCG hardware, or: How I learned to stop worrying and love the bear. Technical Report TR2003-476, Department of Computer Science, Dartmouth College, 2003.

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

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

[23] 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.

[24] T. P. Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In J. Feigenbaum, editor, *Advances in Cryptology – CRYPTO '91*, volume 576 of *LNCS*, pages 129–140. International Association for Cryptologic Research, Springer-Verlag, Berlin Germany, 1992. Extended abstract.

[25] 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.

[26] B. Pfitzmann and M. Waidner. A model for asynchronous reactive systems and its application to secure message transmission. In *Proceedings of the IEEE Symposium on Research in Security and Privacy*, pages 184–200, Oakland, CA, May 2001, IEEE Computer Society Press.

[27] J. Poritz, M. Schunter, E. Van Herreweghen, and M. Waidner. Property attestation—scalable and privacy-friendly security assessment of peer computers. Technical Report RZ 3548, IBM Research, May 2004.

[28] A.-R. Sadeghi and C. Stüble. Taming "trusted computing" by operating system design. In *Information Security Applications*, volume 2908 of *LNCS*, pages 286–302. Springer-Verlag, Berlin Germany, 2003.

[29] A.-R. Sadeghi and C. Stüble. Property-based attestation for computing platforms: Caring about properties, not mechanisms. In *The 2004 New Security Paradigms Workshop*, Virginia Beach, VA, USA, Sept. 2004. ACM SIGSAC, ACM Press.

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

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

[32] R. Sailer, T. Jaeger, E. Valdez, R. Caceres, R. Perez, S. Berger, J. L. Griffin, and L. van Doorn. Building a MAC-based security architecture for the Xen open-source hypervisor. In *ACSAC '05: Proceedings of the 21st Annual Computer Security Applications Conference*, pages 276–285, Washington, DC, USA, 2005. IEEE Computer Society.

[33] R. Sailer, E. Valdez, T. Jaeger, R. Perez, L. van Doorn, J. L. Griffin, and S. Berger. sHype: Secure hypervisor approach to trusted virtualized systems. Research Report RC23511, IBM T.J. Watson Research Center, Yorktown Heights, NY, USA, Feb 2005.

[34] 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, Aug. 2004.

[35] Trusted Computing Group. TPM main specification. Main Specification Version 1.2 rev. 85, Trusted Computing Group, Feb. 2005.

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