Improving the Scalability of Platform Attestation

Frederic Stumpf
Technische Universität Darmstadt
Darmstadt, Germany
stumpf@sec-tud.de

Andreas Fuchs
Fraunhofer Institute for Secure Information Technology
Darmstadt, Germany
afuchs@sit.fraunhofer.de

Stefan Katzenbeisser
Technische Universität Darmstadt
Darmstadt, Germany
skatzenbeisser@acm.org

Claudia Eckert
Technische Universität Darmstadt
Darmstadt, Germany
eckert@sec-tud.de

ABSTRACT

In the process of platform attestation, a Trusted Platform Module is a performance bottleneck, which causes enormous delays if multiple simultaneously attestation requests arrive in a short period of time. In this paper we show how the scalability of platform attestation can be improved. In this context, we propose three protocols that enable fast and secure integrity reporting for servers that have to handle many attestation requests. We implemented all of our protocols and compared them in terms of security and performance. Our proposed protocols enable a highly frequented entity to timely answer incoming attestation requests.

Categories and Subject Descriptors

K.6.5 [**Management of Computing and Information Systems**]: Security and Protection; D.4.6 [**Operating Systems**]: Security and Protection—*Authentication, Invasive software*

General Terms

Security

Keywords

Remote Attestation, Trusted Computing, Performance, Scalability, Attestation Protocols, Integrity Reporting

1. INTRODUCTION

The Trusted Computing Group (TCG) is a non-profit organization that defines open standards for hardware-enabled trusted computing and security technologies. A core component of the specifications issued by the TCG is the Trusted Platform Module (TPM) [**?**], which can be viewed as functionally equivalent to a high-end smart card. The TPM can be used to significantly enhance system security, since it offers means of storing cryptographic keys in a protected manner and is able to establish trust relationships between remote entities [**?**].

To enable the establishment of trust relationships between two different entities, the TPM provides a mechanism to obtain software integrity measurements and to store these measurements in shielded locations called *Platform Configuration Registers* (PCR). These measurements can be reported to a remote entity through the process of platform attestation, which provides a proof of authenticity of the measurements. This process is crucial for ensuring endpoint integrity and thus the trustworthiness of an entity to which confidential data is transferred. Example scenarios that can utilize these concepts include Electronic Commerce, E-Government and Information Rights Management.

The process of remote attestation and the requirements of protocols that support secure integrity reporting have been investigated in the literature [**?**, **?**, **?**, **?**, **?**, **?**]. In this context, all proposed solutions require the TPM to perform one expensive asymmetric cryptographic operation for each entity that is requesting integrity information. One reason for the high complexity is that a verifying entity has to ensure freshness of the integrity information, which is achieved by a challenge-response authentication involving the TPM. However, since the TPM is very limited in its computation power, the computation of one asymmetric operation takes between one and three seconds [**?**]. This causes the process of remote attestation to scale very poorly, which is particularly problematic if an entity is highly frequented and distributes integrity measurements to many clients (such as a server from which all clients request integrity information before they start using a particular service).

In this paper we present protocols that allow to overcome the performance bottleneck of a TPM, so that an entity is able to frequently report its integrity to many clients. To this end, we propose three different protocols that utilize different mechanisms of the TPM: the first protocol extends the most widely used platform attestation by bundling a number of attestation requests and answering them with one TPM operation; the second protocol requires a Trusted Third Party and utilizes hash-chains; and the third protocol realizes integrity reporting using time synchronized tickstamps. Our developed protocols do not require new trusted computing technology or modifications to the TPM specifi-

STC'08, October 31, 2008, Fairfax, Virginia, USA.

cation. We have implemented all of our proposed protocols and have run simulations on currently available hardware TPMs. The simulations clearly indicate that the developed protocols are able to overcome the performance bottleneck of a TPM and can thus be used in environments where an entity is heavily exposed to integrity reporting queries.

The remainder of this paper is organized as follows. In Section **??** we review related work, showing that all existing protocols scale poorly and thus cannot be used in highly frequented environments. Section **??** presents the assumptions and notations for our work. In Section **??** we present our protocols which enable a highly frequented entity to report its integrity. In Section **??** we analyze the security of our proposed solutions and in Section **??** we evaluate the protocols by looking at performance issues. Finally, we conclude in Section **??**.

2. RELATED WORK

Löhr et al. [**?**] present key components for scalable offline attestation. In this context, they introduce an *attestation token* that consists of all necessary information to make a statement whether a specific platform is trusted. This attestation token is public and can thus be shared amongst other entities. However, to verify that a specific platform configuration is trusted and to obtain a proof that a specific attestation token belongs to a specific platform, expensive TPM operations are required. This concept is therefore not applicable in scenarios where frequent integrity verification is needed.

Gasmi et al. [**?**] propose to include integrity information into secure channel establishment. For this purpose, they enhance the TLS-protocol with SKAE certificates [**?**] that additionally carry integrity information. However, their approach requires the computation of multiple expensive asymmetric cryptographic operations on the TPM on both endpoints. This again limits the usefulness in highly frequented environments due to massive performance degradations.

Shi et al. [**?**] address the time-of-use and time-of-attestation discrepancy of the TPM-based binary attestation. This discrepancy emerges because a software component is measured before execution and not directly before attestation. To overcome this problem, the authors propose to only attest to a small piece of sensitive code and thus measure a particular piece of code immediately before execution. However, the work requires that for every established communication channel, two `TPM_Seal`, one `TPM_Unseal` and one `TPM_Sign` operation are computed on the TPM, thus rendering the approach inapplicable in a client-server scenario.

Another approach to overcoming the bottleneck of a TPM is to use virtual TPMs [**?**]. A virtual TPM is a software TPM that only uses the underlying hardware TPM for certain operations. This approach significantly increases the performance of the attestation process, since the CPU is used to calculate the attestation related operations. On the downside, because the TPM is implemented in software, this approach does not offer the same security as a hardware TPM. In particular, a software TPM does not provide functions such as active shields or active security sensors for preventing unauthorized access.

Another proposal [**?**] considers extending a TPM with hardware-based virtualization techniques and attaching the TPM to a faster BUS or integrating it directly into the CPU. Since the TPM possesses much more computation power in this approach, performance degradation in the attestation process does not occur. However, the approach requires modifications of the TPM architecture.

Our work uses the TPM-based binary attestation which requires a trusted OS that performs measurements of all executed code, i.e., binary attestation. In contrast, [**?**] and [**?**] focus on semantic attestation based on attesting the behavior of software components. However, semantic attestation also requires a TCG-enhanced boot process to ensure that a small operating system kernel applies mechanisms that enforce the semantic attestation. To this end, a partially TPM-based binary attestation is required so that time degradations of the attestation process also apply to approaches based on semantic attestation.

Other related work, such as [**?**, **?**, **?**] additionally require that the communication perform expensive TPM operation during the execution of the attestation protocol.

3. ASSUMPTIONS AND NOTATIONS

In the following, we call the entity who wants to attest to the contents of its platform configuration the server and we call the entities that require an attestation the clients. The server is also referred to as prover, since he wants to prove to the verifier (a client) that he is in a trusted system configuration.

We assume that a trusted operating system, that is either based on a virtual machine monitor [**?**, **?**, **?**] or on a microkernel [**?**, **?**, **?**], performs integrity measurements on running software. However, specifying and presenting the operating system environment is beyond the scope of this paper.

We use the following notation: C_x denotes a client with $x \in \{1, \ldots, n\}$ and S denotes a server; Na_x and Nb_x are random number nonces that are used to verify freshness and to detect impersonation. We denote a set of platform configuration registers as *PCR* and the Stored Measurement Log as *SML*.

4. SCALABLE SOLUTIONS

In this section we will first highlight the shortcomings of the integrity reporting mechanisms as defined by the TCG [**?**]. We will then show how attestation can be used in highly-frequented scenarios. In this context, we will present three different solutions that overcome the performance bottleneck of a TPM.

Figure **??** shows the delays that occur when multiple attestation requests arrive simultaneously at one server. Each attestation request includes one *nonce* that is generated by the challenger, to ensure freshness of the attestation reply. Every request can not be answerred until the TPM has performed the `TPM_Quote` command, which signs a number of platform configuration registers and the nonce of the requester with an *Attestation Identity Key* (AIK). While the TPM performs internal computation, it is locked and does not accept any other request from the software stack. Thus, attestation requests are non-parallelizable. Incoming attestation requests are delayed until the TPM is able to process a new `TPM_Quote` command. We have implemented the integrity reporting protocol specified by the TCG and performed measurements on currently available TPMs (see Section **??**). Our simulations on different hardware TPMs showed that a single `TPM_Quote` operation takes about one second, which verify the results presented in [**?**] stating that

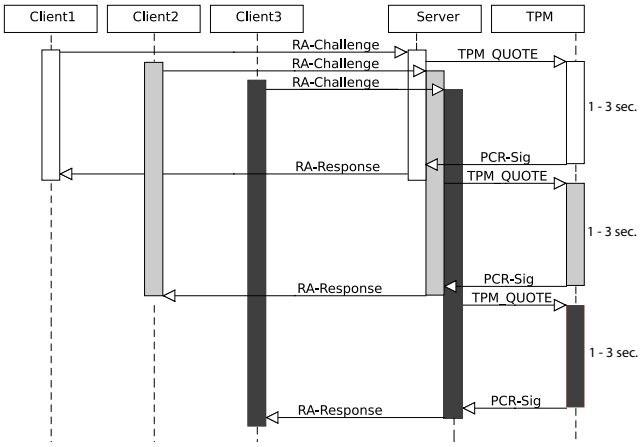

Figure 1: Delay of simultaneous arriving attestation requests

the time required for computing one `TPM_Sign` operation is between one and three seconds.

Figure **??** points out that even if a small number of clients require an attestation, massive delays will occur due to the sequential operation of the TPM. That is not only the case when the `TPM_Quote` command is used, but also when integrity reporting is realized implicitly through other TPM-operations, e.g., through *sealing* [**?**] or *binding* [**?**, **?**]. We also ran simulations of an attestation protocol that utilizes `TPM_CertifyKey` combined with `TPM_Unbind` as proposed by Löhr et al. [**?**] and Sadeghi et al. [**?**]. These protocols enable platform attestation by sealing a non-migratable TPM key to a set of platform configuration registers. However, the time for computing one `TPM_Unbind` operation (around one second[1]) is comparable to computing one `TPM_Quote` operation. These results clearly indicate that computing asymmetric cryptography on a TPM is very expensive, making currently proposed attestation protocols hardly applicable in highly-frequented scenarios.

4.1 Multiple-Hash Attestation

The Multiple-Hash Attestation protocol extends the integrity reporting defined by the TCG by bundling a number of attestation requests and answering them with one `TPM_Quote` operation. For this purpose, we utilize the properties of a collision resistant hash function. In particular, we map a number of incoming attestation requests to a single request whose nonce is computed as a hash of all nonces of the arriving attestation requests. Before the nonces are passed to the TPM, they are also added to a *NonceList* that describes which nonces were hashed to produce the target nonce. The prover then transmits the output of the `TPM_Quote` command together with the *NonceList* and the *SML* to all entities that issued an attestation request. Before accepting the proof, the verifier checks whether his nonce is part of the *NonceList* transmitted alongside the `TPM_Quote` output.

To prevent masquerading attacks on the authenticity of the platform configuration [**?**], it is necessary to integrate an authentication process in the attestation protocol. These attacks forward the integrity measurements of a conform host to masquerade a conform system state. Therefore, our proposed protocols add a key-establishment in order to ensure that the channel of attestation is authentic. The negotiated key can then used as an encryption key for all subsequent messages sent between server and client. This mechanism also guarantees an end-to-end communication and keeps the attestation channel from becoming compromised by another application that could take over the attestation channel after the attestation has succeeded. Protocol **??** shows the resulting protocol flow.

[1]All measurements were performed on a TPM ST Microelectronics ST19WP18-TPM-C and an Atmel TPM 1.2

Protocol 4.1.1: Multiple-Hash Attestation

---

1. *Precomputation by S.* S selects an appropriate prime p and generator g of Z_p^* $(2 \leq g \leq p-2)$. S chooses a random secret s, $2 \leq s \leq p-2$, and computes $g^s \bmod p$. S transmits p and g to all C_x.

2. *Precomputation by C_x.* C_x chooses a random secret c_x, $2 \leq c_x \leq p-2$, and computes $g^{c_x} \bmod p$.

3. *Attestation challenge.* C_x, $x = 1, \ldots, n$, choose a non-predictable nonce Na_x and transmit message (**??**) to S.

$$C_x \rightarrow S: \quad Na_x,\ g^{c_x} \bmod p \tag{1}$$

4. S adds $Na_1, \ldots, Na_n$ to the *NonceList* and computes:

$$NonceList = Na_1 \,||\, Na_2 \,||\, \ldots \,||\, Na_n \tag{2}$$

$$\begin{aligned} HashedNonceList = h(h(Na_1, g^s \bmod p) \,||\, \\ h(Na_2, g^s \bmod p) \,||\, \ldots \,||\, h(Na_n, g^s \bmod p)) \end{aligned} \tag{3}$$

5. S then computes the attestation response message by signing the *HashedNonceList* and a set of *PCRs* using an *AIK*. S then transmits message (**??**) to C_x.

$$\begin{aligned} S \rightarrow C_x: \quad & Cert(AIK),\ NonceList,\ g^s \bmod p, \\ & \{HashedNonceList,\ PCR\}_{K_{AIK}^{-1}} \end{aligned} \tag{4}$$

6. *Key confirmation.* C_x computes the shared session key by computing $K_{SC_x} = (g^s)^{c_x} \bmod p$. C_x then generates a second non-predictable nonce (Nb_x) and transfers message (**??**) to S:

$$C_x \rightarrow S: \quad \{Nb_x,\ g^{c_x} \bmod p\}_{K_{SC_x}} \tag{5}$$

7. S computes the shared session key by computing $K_{SC_x} = (g^{c_x})^s \bmod p$ and decrypts the received message with K_{SC_x}. S then transfers message (**??**) to C_x.

$$S \rightarrow C_x: \quad \{Na_x,\ Nb_x,\ SML,\ g^s \bmod p\}_{K_{SC_x}} \tag{6}$$

8. C_x verifies the signature of $\{HashedNonceList,\ PCR\}_{K_{AIK}^{-1}}$ and checks the freshness of Na_x, by recalculating *HashedNonceList* using the transmitted *NonceList*. Based on the received *SML* and the *PCR* values C_x processes the *SML* and re-computes the received *PCR* values. If the computed values match the signed *PCR* values, the *SML* is valid and untampered. Finally, C_x has to verify that the delivered integrity reporting values match the given reference values; thus C_x can decide if S is in a trusted system state.

The above solution works well if several attestation requests arrive at once; if they are slightly delayed, the Multiple-Hash Attestation Protocol can be optimized further. For this purpose, we introduce a ring buffer with three different areas, each consisting of one area for the *NonceList* and another area for the output of a TPM operation. Figure **??** depicts this tri-state ring buffer and shows how it is embedded in the attestation protocol. This tri-state ring buffer holds all relevant data for three different threads that are responsible for passing data from the verifier to the TPM and vice versa:

- Input thread: This thread collects challenge-requests and writes them to a *NonceList* area of the ring buffer.
- Working thread: This thread computes the *HashedNonceList* from the *NonceList* stored in one of the three areas of the buffer and executes the `TPM_Quote` on the *HashedNonceList* and the *PCR*. The result is then written back to the same are of the buffer and the ring buffer is rotated.
- Output thread: This thread is responsible for reading the results of the TPM operation from the buffer and for delivering the results to the corresponding clients who requested attestation.

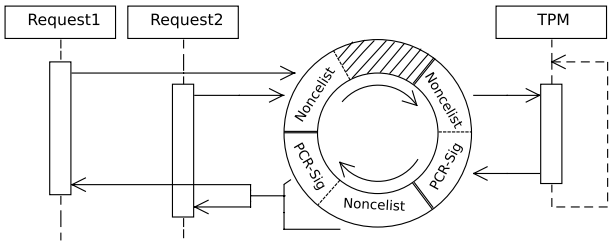

Figure 2: Tri-state ring buffer of the Multiple-Hash Attestation

Each time a new attestation request arrives at the server, the input thread forks a new process. This process adds the incoming nonce to the *NonceList* and remembers the area of the ring buffer that is responsible for the process. The working thread runs continuously; this thread is responsible for passing data to and receiving data from the TPM. As soon as the working thread receives an answer from the TPM, the working thread writes the results in the ring buffer and rotates the tri-state ring buffer. This working thread then retrieves the new *NonceList* of the adjacent area, which he hashes to create the *HashedNonceList* and passes to the TPM. While the working thread is operating on the TPM, the processes that are responsible for answering the client's attestation-requests are waiting until the tri-state buffer has been rotated one round and their data is ready. As soon as the requested data is present in the buffer, the processes that are responsible for the attestation-request can deliver the attestation-response, consisting of the signed *HashedNonceList* and *PCR*, to the requester.

This approach enables a very efficient realization of the Multiple-Hash Attestation. Since the working thread that is responsible for the TPM is also responsible for the ring buffer, a maximum utilization of the TPM can be achieved. Incoming attestation requests must wait a maximum of two rotation steps of the ring buffer before the result is available.

4.2 Timestamped Hash-Chain Attestation

Another alternative to reduce the number of costly TPM operations, is to relinquish the server from integrating every nonce of each client into the costly TPM operations. This can be achieved by involving a Trusted Third Party (TTP) that is responsible for issuing nonces for the attestation process. For this purpose, the TTP provides nonces, signatures of these nonces and timestamps that state when the TTP generated the nonces. We divide the time into intervals; each nonce together with the timestamp is associated to a particular time interval. To ensure an attestation request arrives at the server in a particular interval, the server uses the nonce which is valid in the current interval and not a nonce provided by the client. The TTP and all clients need to be loosely time-synchronized to enable the clients to verify the validity of the TTP generated nonce for a certain interval.

A straightforward implementation of this idea results in a high load at the TTP. During i intervals, the TTP has to generate $n * i$ nonces and timestamps to serve n servers. Since the intervals should be in the range of seconds, a TTP has to generate new nonces very frequently. To releave the TTP from generating a nonce for each interval, we propose using hash-chains [**?**]. The TTP issues a nonce Na_0 with time-stamp only for the first time interval and the server uses the initial TTP-generated nonce Na_0 only for the first attestation query. After each interval the server performs a hash-operation on the nonce by applying the hash function h successively to the nonce of the previous interval in order to produce the nonce of the next interval, i.e., $Na_{v+1} = h(Na_v)$, with $v = 0, 1, \ldots, k$.

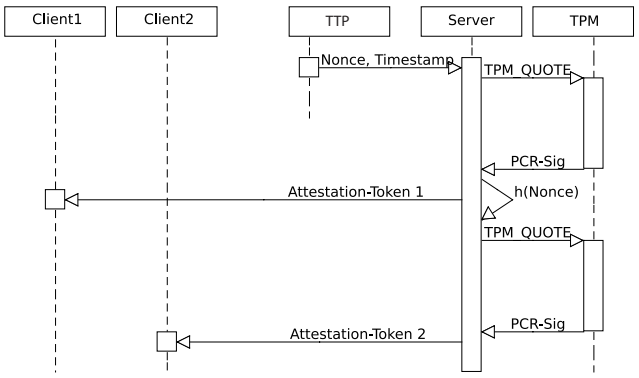

Figure 3: Protocol flow of the Timestamped Hash-Chain Attestation

Since the TTP issues a timestamp only for the first nonce, the verifying clients cannot directly validate whether the received nonce is valid in the current interval. To enable the clients to verify the validity of the nonce, the server has to provide a proof that he applied the hash function the correct number of times. Since the attestation service of the server is part of the server's platform configuration and thus its state is included in the *SML*, the clients can verify that the attestation service is in a trusted state. The proof is thus being made through validating the platform configuration of

the service.

Figure **??** depicts the Timestamped Hash-Chain Attestation. After each interval, the server calculates a new hash-value with which the new attestation is performed. To enable the clients to validate the freshness of the platform configuration, the server delivers an attestation token (τ) to the requester. This attestation token consists of the TTP-signed seed nonce together with a timestamp, an *AIK* signed message with the current interval count, the nonce of the current interval, the *PCRs* and the public Diffie-Helman key of the server and the certificate of the *AIK*. The attestation token therefore consists of all information that is necessary to validate the freshness of the attestation response:

$$\begin{aligned}\tau \quad = \quad & \{Na_0,\ time_0\}_{K_{TTP}^{-1}}, Cert(AIK) \\ & \{h^v(Na_0),\ v,\ PCR,\ g^s \bmod p\}_{K_{AIK}^{-1}}\end{aligned} \tag{7}$$

The attestation token introduced here has similarities to the one introduced in [**?**]. However, the main difference is that a verifier can validate the platform configuration without requiring the prover to perform expensive TPM operations. To verify that the platform configuration of the server is trusted, the clients have to verify the validity of all certificates, all signatures and the validity of the timestamp. Furthermore, the clients have to verify whether the received nonce is valid in the current active interval. For this purpose, the following equation must hold: $time_0 + v \cdot t \leq now \leq time_0 + (v+1) \cdot t + \epsilon$, where v represents the interval number, t the time length of the interval, and *time* the point of time when the TTP generated the *nonce*. Moreover, *now* denotes the current time of the verifier and ϵ a certain error range. The time length of the interval is part of the *SML* and thus represented by the platform configuration of the server. The resulting protocol is shown in Protocol **??**.

Protocol 4.2.1: Timestamped Hash-Chain Attestation

---

1. S chooses a TTP for providing the seed nonce and the TTP transfers message (**??**) to S.

$$TTP \rightarrow S: \quad \{Na_0,\ time_0\}_{K_{TTP}^{-1}} \tag{1}$$

2. *Precomputation by S.* S selects an appropriate prime p and generator g of Z_p^* $(2 \leq g \leq p-2)$. S chooses a random secret s, $2 \leq s \leq p-2$, and computes $g^s \bmod p$. S transmits p and g to all C_x.

3. *Precomputation by* C_x. C_x chooses a random secret c_x, $2 \leq c_x \leq p-2$, and computes $g^{c_x} \bmod p$.

4. *Attestation challenge.* C_x, $x = 1, \ldots, n$, deliver message (**??**) to S.

$$C_x \rightarrow S: \quad g^{c_x} \bmod p \tag{2}$$

5. Depending on the current interval v, S computes the current valid nonce Na_v by calculating:

$$Na_v = h(Na_{v-1}), \quad \text{with} \quad v = 1, \ldots, k \tag{3}$$

6. S computes the attestation response message by signing Na_v, the current interval v and the set of requested *PCRs* using an *AIK* thereby obtaining $\{h^v(Na_0), v, g^s \bmod p, PCR\}_{K_{AIK}^{-1}}$.

7. S transmits the attestation token τ in message (**??**) to C_x.

$$\begin{aligned}S \rightarrow C_x: \quad & \{Na_0,\ time_0\}_{K_{TTP}^{-1}},\ Cert(AIK), \\ & \{h^v(Na_0),\ v,\ g^s \bmod p,\ PCR\}_{K_{AIK}^{-1}}\end{aligned} \tag{4}$$

8. *Key confirmation.* C_x computes the shared session key by computing $K_{SC_x} = (g^s)^{c_x} \bmod p$. C_x then generates a second non-predictable nonce (Nb_x) and transfers message (**??**) to S.

$$C_x \rightarrow S: \quad \{Nb_x,\ g^{c_x} \bmod p\}_{K_{SC_x}} \tag{5}$$

9. S computes the shared session key by computing $K_{SC_x} = (g^{c_x})^s \bmod p$ and decrypts the received message with K_{SC_x}. S then transfers message (**??**) to C_x.

$$S \rightarrow C_x: \quad \{Na_v,\ Nb_x,\ SML,\ g^s \bmod p\}_{K_{SC_x}} \tag{6}$$

10. C_x verifies all signatures and checks the freshness of the Na_v by checking whether Equations (**??**) and (**??**) hold:

$$time_0 + v \cdot t \leq now \leq time_0 + (v+1) \cdot t + \epsilon \tag{7}$$

$$h^v(Na_0) \stackrel{?}{=} Na_v \tag{8}$$

11. Finally, C_x verifies that the platform configuration of S is trusted based on the *SML* and the *PCR*s.

4.3 Tickstamp Attestation

The tickstamp attestation uses the tick-counter of a TPM (a tick is in the time range of milliseconds up to seconds). A TPM provides a mechanism to create a signature of the current tick-counter value. The resulting data structure includes a signature of the current tick-counter value and the time interval after which the counter is periodically incremented.

To use tick-counters during platform attestation, we utilize the concept that a TPM enables the creation of non-migratable keys that are bound to the platform configuration. Before a TPM performs platform attestation, a non-migratable key is generated on the TPM and bound to a specific set of platform configuration registers. This key is certified using an *AIK*, which gives a proof that the key is bound to a specific set of platform configuration registers. The bound key is then used in periodic time intervals to generate *TickStampBlobs*, which is only possible while the platform configuration has not changed. A *TickStampBlob* consists of a complete `TPM_CURRENT_TICKS` structure [**?**]

and the resulting signature. It includes the current value of the tick-counter, a tick-session identifier, and a signature of the data. To demonstrate to a remote party that the platform configuration is to be trusted, an *attestation token* τ is computed. This token comprises the latest *TickStampBlob*, which is the first part of the token shown in Equation (**??**), as well as the certificate of the key used to generate the *TickStampBlob*:

$$\tau = \{currentTicks,\ g^s \bmod p\}_{K_{TS}^{-1}},\ Cert(K_{TS}),\ Cert(AIK) \quad (9)$$

The trustworthiness of the platform configuration can be evaluated based on the certificate and the tick-count value inside the *TickStampBlob*. However, the token gives only an assertion about the platform configuration in relation to the tick-counter on the platform that wants to demonstrate its configuration. A synchronization of the tick-counter of the challenger and the prover is thus needed.

In the following, we will discuss two means of achieving this synchronization. Most implementations of the TPM specification initialize a tick-session after a reboot of the system with the value zero. To uniquely identify a specific tick-session, the TPM also adds a nonce to the tick-session. Since the nonce is also part of every *TickStampBlob*, two different *TickStampBlob* structures can be uniquely related to their session.

Challenge-Response Synchronization.

The easiest way to perform synchronization with the tick-session of the server is to deliver a nonce to the server which is then signed with `TPM_TickStampBlob`. The resulting signature includes the complete `TPM_CURRENT_TICKS` structure [**?**], which gives an assertion about the actual tick-counter value, the tick-rate and the identifying nonce of the current tick-session. Based on this information, the verifier can check the freshness of the actual attestation token. However, this concept requires one expensive sign operation on the TPM, which does not scale. It is thus not applicable in highly-frequented environments.

Time Synchronization using a TTP.

Another alternative is to involve a Trusted Third Party in the synchronization protocol. This sync-TTP is responsible for associating a specific tick-session to a specific global time. For this purpose, the TTP generates and transfers a nonce to the server, directly after the tick-session on the server is initialized. This nonce is then signed with `TPM_TickStampBlob` by the server TPM and the result is delivered to the sync-TTP. The sync-TTP verifies the signatures and generates a synchronization token that includes the global time, the round-trip time and the received result. This sync token is then returned to the server which adds the token to all generated attestation tokens.

Based on the synchronization token, an association between the global time and the beginning of a specific tick-session is made. A client only needs to synchronize his time with the sync-TTP in order to make a statement about the freshness of the received attestation token τ in Equation (**??**). The sync-TTP thus provides services similar to a generic NTP server. It is also reasonable that a NTP server can be extended with the ability to create synchronization tokens, since these protocol steps only require minimal additional computations. Tickstamp Attestation with TTP-based time synchronization is shown in Protocol **??**.

Protocol 4.3.1: Tickstamp Attestation with synchronization token

---

1. S selects a TTP for providing a sync-token. The TTP transfers a nonce N_t using message (I) to S

$$TTP \rightarrow S: \quad Nt \quad (I)$$

2. S creates a non-migratable TPM key (K_{TS}) that is bound to a specific set of platform configuration registers. S certifies K_{TS} with K_{AIK}^{-1}. The resulting structure is denoted as $Cert(K_{TS})$ and gives an assertion to which *PCRs* K_{TS} is bound. S then signs the actual tick-counter value with K_{TS}^{-1} and delivers message (II) to the *TTP*.

$$S \rightarrow TTP: \quad \{Nt,\ currentTicks\}_{K_{TS}^{-1}},\ Cert(K_{TS}),\ Cert(K_{AIK}) \quad (II)$$

3. The *TTP* verifies the signature and creates a time stamp on the received message and transfers the sync-token (τ_{sync}) in message (III) to S.

$$TTP \rightarrow S: \quad \{\{Nt,\ currentTicks\}_{K_{TS}^{-1}},\ Time\}_{K_{TTP}^{-1}} \quad (III)$$

The server has now received a sync-token that can subsequently be used by the clients to verify freshness of the attestation token. After completing this initialization phase, the server is ready to answer attestation requests.

1. *Precomputation and Pre-deployment by S.* S selects an appropriate prime p and generator g of Z_p^* ($2 \leq g \leq p-2$). S chooses a random secret s, $2 \leq s \leq p-2$, and computes $g^s \bmod p$. S transmits p and g to all C_x.

2. *Precomputation by* C_x. C_x chooses a random secret c_x, $\quad 2 \leq c_x \leq p-2$, and computes $g^{c_x} \bmod p$.

3. *Attestation challenge.* $C_x, x = 1, \ldots, n$, deliver message (**??**) to S.

$$C_x \rightarrow S: \quad g^{c_x} \bmod p \quad (1)$$

4. The server periodically signs the actual tick-counter value with K_{TS}^{-1}. The resulting data structure is denoted as *TickStampBlob*.

5. S transmits the synchronization token $\tau_{sync} = \{\{Nt,\ currentTicks\}_{K_{TS}^{-1}},\ Time\}_{K_{TTP}^{-1}}$ and the attestation token τ in message (**??**) to C_x.

$$S \rightarrow C_x: \quad \{currentTicks, g^s \bmod p\}_{K_{TS}^{-1}},\ Cert(K_{TS}), Cert(AIK), \tau_{sync} \quad (2)$$

6. *Key confirmation.* C_x computes the shared session key by computing $K_{SC_x} = (g^s)^{c_x} \bmod p$. C_x then generates a second non-predictable nonce (Nb_x) and transfers message (**??**) to S.

$$C_x \rightarrow S: \quad \{Nb_x, g^{c_x} \bmod p\}_{K_{SC_x}} \quad (3)$$

7. S computes the shared session key by computing $K_{SC_x} = (g^{c_x})^s \bmod p$ and decrypts the received message with K_{SC_x}. S then transfers message (**??**) to C_x.

$$S \rightarrow C_x : \quad \{Nb_x, SML, g^s \bmod p\}_{K_{SC_x}} \qquad (4)$$

8. Finally, C_x verifies all signatures and checks whether the attestation token τ is fresh using the synchronization token τ_{sync}. In addition, C_x verifies that the platform configuration of S is trusted based on the SML and the PCRs.

5. SECURITY ANALYSIS

In this section we will discuss the security of our proposed protocols. Since all protocols include an authentication step, we will first analyze whether this step is secure and enables secure integrity reporting. We finally perform a formal verification using the AVISPA protocol prover [**?**].

5.1 Security of the Authentication

In this section we will discuss the security of the authentication step by looking at potential attacks, including man-in-the-middle and version rollback attacks.

5.1.1 Man-in-the-Middle attack

All presented protocols prevent an attacker from hiding his malicious software configuration by performing a relay attack [**?**, **?**], since all subsequent messages are encrypted with the computed session key K_{SC_x}. It is not possible for an attacker to perform some sort of man-in-the-middle attack and to establish two different cryptographic sessions between the verifier and the challenger, as he is not able to modify the attestation response of the prover. Since the session key K_{SC_x} is protected by the trusted operating system (e.g., by storing it in a special purpose region of a security kernel or in a special virtual machine) it is not possible to extract this key under normal run-time conditions. Changing the trusted operating system environment to enable extraction of this key would lead to a non-conformant system state which would have been detected in the attestation phase.

5.1.2 Version-rollback attack

The presented protocol is not compatible with a verifier that expects the insecure existing remote attestation defined by the TCG. This is especially true for the Multiple-Hash Attestation. This attack was misleadingly classified in [**?**] as a man-in-the-middle attack. However, this classification is insufficient, since the attack requires that one entity executes the insecure integrity reporting protocol specified by the TCG in [**?**].

In this attack scenario, three different parties, a verifier, a prover and an adversary, are involved. The adversary tries to relay the attestation challenge of the verifier to the prover, thus trying to masquerade a trustworthy system configuration. The verifier and the adversary run an authentication enhanced attestation protocol (e.g., Protocol **??**), while the prover runs the TCG-defined attestation protocol. This attack can be classified as a version rollback attack, in which the adversary masks his public key together with the nonce provided by the verifier as new nonce for the prover. The prover does not verify the syntax of the attestation challenge. He directly signs the masqueraded nonce including public key and delivers the computed signature to the adversary. The adversary simply forwards the obtained message and both adversary and verifier compute the shared key based on the exchanged public keys. However, since the protocol's software integrity of the prover is also reflected in the platform configuration, the verifier will determine the platform configuration as not being trusted. The version rollback attack is therefore only a theoretical attack and fails, since it is not possible to successfully masquerade a trusted system configuration.

5.2 Formal Security Analysis

To formally analyze the proposed protocol, we use the AVISPA protocol prover [**?**]. AVISPA provides a special language [**?**, **?**] for describing security protocols and specifying their intended security properties (we refer to the AVISPA website for more details[2]).

AVISPA is a powerful tool to formally analyze cryptographic protocols. Since our proposed protocols include an authentication phase, it is necessary to analyze whether the protocols are secure against Dolev-Yao attackers [**?**].

In the following we restrict us to analysis of Protocol **??**. All other protocols can be handled analogously. Each involved entity (i.e., prover and verifier), is modelled as a finite state machine and each transition from one state to another requires the receipt of a message and the sending of a reply message. Thus, the verifier is modelled as a state machine with three states and the prover is modelled as a state machine with two states. In each state, the respective message as specified in the protocols are transferred, e.g., the prover receives in its first state message (4) of Protocol **??** and creates and delivers message (5) to the verifier.

We use the Dolev-Yao intruder model [**?**] to model the attacker and the environment. In this intruder model the attacker has full control over all messages that are sent over the network. The attacker can therefore intercept, analyze or modify messages, as well as compose new messages and send the messages to any party. To abstract from the negotiation of a common generator g and a common group Z_p^*, we assume that these are global parameters known to all parties in the environment. However, this is not a security restriction since these messages can transferred in plaintext without loss of security (see [**?**]).

We use AVISPA to verify the following security goals:

- C_x authenticates a genuine and authentic TPM on the value Na_x. This holds since only an authentic TPM is able to sign Na_x with a corresponding non-migratable key (AIK or special purpose key).
- C_x authenticates the TPM of S on the value K_{SC_x}. This holds since given the first statement, only the owner of the TPM possesses the corresponding private Diffie-Helman key. As a consequence, only C_x is able to compute the secret key based on the provided public key.
- C_x authenticates the TPM of S on the value Nb_x. This holds since given the second statement, only S can decrypt Nb_x and send it back to C_x.

[2] http://www.avispa-project.org

- C_x and S share the key K_{SC_x}, which is confidential and should be kept secret.
- C_x and S share the Stored Measurement Log (SML), which is privacy related and should also be kept secret.

It should be noted that we do not directly authenticate S to C_x. C_x only determine whether they are currently communicating with the platform that has provided authentic measurements and that this channel is authentic.

After modelling the protocol, we analyzed the model with the model checker provided by AVISPA. We found no attack trace; thus the protocol analyzer reports that all security properties are satisfied and Protocol **??** is secure.

5.3 Security Considerations of the Multiple-Hash Attestation

The main difference between the existing TCG-defined integrity reporting and the Multiple-Hash Attestation is that multiple nonces are hashed to one single nonce. The security of this process relies on the property that the hash function, in our case SHA-1, is collision resistant. If a collision resistant hash function is used, it is infeasible for an adversary to find a collision that can be used to masquerade a trustworthy system configuration.

5.4 Security Considerations of the Timestamped Hash-Chain Attestation

A general problem in the process of platform attestation is that an unfresh platform configuration can be replayed, leading to a non-trustworthy platform configuration being masqueraded as trustworthy. This attack is prevented by using randomly generated nonces combined with the challenge-response authentication method. However, in the context of the Timestamped Hash-Chain Attestation, these nonces are derived from one seed nonce by applying a hash function on this value. It is therefore possible to generate nonces that are valid in the future. This property can be exploited by an adversary by generating nonces that are valid in future intervals, computing the attestation token in a trustworthy configuration, and replaying the computed attestation token in the future after compromising the platform.

The risk that an adversary may perform such an attack can be minimized by preventing an adversary's ability to inject nonces corresponding to future time intervals. That can be done, for example, by modifying the operating system so that it only allows certain trusted processes to communicate with a TPM. These trusted processes should only accept seed nonces with a valid signature that have been created by a TTP. Since the configuration of the operating system is also part of the platform configuration, a verifier can check whether the prover's OS is in a trusted state and thus possesses a mechanism to prevent attacks of this type.

5.5 Security Considerations of the Tickstamp Attestation

The security of the Tickstamp Attestation relies on the assurance that a specific non-migratable TPM key, satisfying certain criteria, is used. This assurance is made using a certificate generated through `TPM_CertifyKey`. Löhr et al. [**?**] verified that the concept of binding a key to a specific set of platform configuration registers is secure against man-in-the-middle attacks. We will thus only look at the differences between the protocol proposed in [**?**] and our proposal. In contrast to Löhr et al., we also integrate a public Diffie-Helman key into the K_{TS} signed message. Using K_{TS} as a signing key at a specific time is only possible if the platform configuration is in a known and trusted state. The extraction of the Diffie-Helman key requires a modified system configuration that causes the state of the platform to change. The TPM will then deny decryption of the sealed key K_{TS}. To further enhance security, the Diffie-Helman key should also be held in a special purpose region of a microkernel or virtual machine, as, for example, proposed in [**?**]. It should also be noted that each time K_{TS} is used, it is verified that the actual platform configuration is consistent with the platform configuration K_{TS} was bound to.

6. EVALUATION

In this section we will first present the performance measurements of our implementation. We will then perform a comparison of our proposed protocols.

6.1 Performance Evaluation

The main goal of our proposal is to enhance the scalability of platform attestation. We implemented all of our proposed protocols in Java using the tpm4java framework [3] and ran performance simulations. The advantage of this framework is that it is a very efficient implementation and we can talk directly to the */dev/tpm* device driver without requiring that another TPM software stack be present in the system. A software stack in the background would additionally need computation power and thus decrease performance. Our approach causes the time degradation to depend only on the TPM. We measured the runtime of all protocols, excluding those for generating the Diffie-Helman keys and those for generating a new TPM non-migratable key (K_{TS}) required in Protocol **??**. We performed our measurements on a Core2Duo E6700 with 2.66GHz, 2GB RAM running OpenSuSE 10.3 and an Atmel TPM 1.2. The results of our measurements are depicted in Figure **??**.

The results shown in Figure **??** were obtained by averaging over 100 independent runs for each protocol. For each protocol, we measured the latencies when one attestation challenge arrives, as well as when multiple (100) attestation challenges arrive simultaneously. The latencies of the TCG-defined protocol [**?**] as well as the Robust-IRP [**?**] scale linearly. Therefore, the time for answering n simultaneously arriving attestation request is approximately equal to $n * x$, where x is the time for answering one attestation.

The average column also depicts the time that is necessary to execute the key confirmation phase once the TPM has delivered the integrity information; this time is shown as the second summand in the column. Note that the TCG-defined IRP in Figure **??** does not has a second summand as the TCG-defined protocol does not require a key-establishment.

The time to finish a Multiple-Hash Attestation varies roughly between 1 and 2 seconds, this variation is caused by the ring buffer. If the ring buffer has just been rotated one step and new requests arrive, these requests have to wait until the TPM has finished calculation and the buffer is rotated again. The Tickstamp-Attestation and the Timestamp-Attestation also consider the time that is needed to generate the attestation token. As soon as the token has been generated, it can be used to attest to the contents of

[3] http://tpm4java.datenzone.de

Protocol	No. concurrent Req.	Average (ms)	Min (ms)	Max (ms)
TCG-IRP [?]	1	852, 47	842	879
	n	852,47 · n		
Robust-IRP [?]	1	863,75 + 3,4	860	881
	n	869,75 · n + 3,4		
Multiple-Hash Attestation	1	939,91 + 6,16	921	973
	n (Best)	936,61 + 6,68	892 + 1	
	n (Worst)	1826,84 + 6,68		1826,84 + 20
Timestamped Attestation	Token generation time	1448,95	1422	1727
	1	<1 + 6,36		
	n	<1 + 6,36		
Tickstamp Attestation	Token generation time	1456,58	1424	1686
	1	<1 + 6,42		
	n	<1 + 6,42		

Figure 4: Measured latencies of selected Integrity Reporting Protocols

the platform configuration registers. This token generation time therefore indicates the length of the attestation interval in which this attestation token is used. An attestation interval can, therefore, not be smaller than about 1.5 seconds. The measurements show that all proposed protocols are independent of the number of simultaneously arriving requests. They can therefore significantly reduce the time required to answer simultaneously arriving attestation requests.

6.2 Comparison of the protocols

All proposed protocols enable a highly-frequented server to timely answer all incoming attestation requests. While the Multiple-Hash Attestation is the slowest and requires roughly between one and two seconds to complete the attestation process, it has the advantage that it is very similar to the existing TCG-proposed integrity reporting protocol. Prover and verifier must, therefore, only minimally modify their attestation interface. This protocol can be classified as an *active attestation*, since it requires the verifier to provide a nonce for the server. To use the protocol a direct connection to the verifier must be established; it is thus not possible to relay the attestation message to other parties.

The biggest advantage of Timestamped Hash-Chain Attestation and the Tickstamp Attestation is that these protocols are *passive attestations*, which require no direct communication between server and client to deliver integrity information. A client can thus collect attestation tokens which he received from other entities and see how the configuration of a particular server changed over time. If this is a desirable feature, the SML must be integrated inside the attestation tokens τ, which would remove the need of a direct connection to the prover and the verifier can collect attestation tokens without noticing the server. However, to ensure freshness of the attestation token and to complete the authentication process, a direct communication between server and client is needed.

Both protocols require a TTP either for providing a synchronization token or for the initial nonce. Since the Timestamped Hash-Chain Attestation requires additional security mechanisms, we suggest to use, depending on the scenario, either the Tickstamp Attestation or the Multiple-Hash Attestation.

To prevent an attacker from performing masquerading attacks on the authenticity of the platform configuration, our protocols establish a secure channel between the involved entities. This secure channel is then used for transmitting the privacy-related Stored Measurement Log. Although the computation of these cryptographic operations degrade the server performance, we believe that ensuring the authenticity of integrity information is necessary in order to enable secure integrity reporting.

However, it should also be noted that these cryptographic operations can be computed very efficiently and only increase the time for answering one attestation request by 0.70% and are in the range of milliseconds.

7. CONCLUSION

In scenarios where an entity is frequently requested to deliver integrity information, existing protocols for performing the TCG-specified platform attestation scale badly. Such scenarios mainly include a client-server architecture, where a large number of clients frequently request integrity information of a particular server. This restriction is caused by the fact that a Trusted Platform Module (TPM) possesses very restricted computation power and is highly involved in the process of platform attestation. In this paper we proposed three solutions to overcome the bottleneck of a TPM and thus to perform platform attestation in scenarios where a frequent integrity verification is needed. Our proposed protocols do not require any modifications to the TPM hardware or any modifications to the measurement process. Although our protocols are based on the TPM-based binary attestation and treat PCR values as measurement data, they could be easily modified to send other forms of measurement data, such as measurement data collected in run-time measurement systems. We presented a performance evaluation as well as a security analysis of our proposed protocol; the results clearly indicate that the protocols can considerably reduce the performance overhead of the attestation process.

8. REFERENCES

[1] AVISPA. Deliverable 2.3: The high-level protocol specification language. Technical report, `http://www.avispa-project.org/delivs/2.1/d2-1.pdf`, 2003.

[2] S. Berger, R. Cáceres, K. A. Goldman, R. Perez, R. Sailer, and L. van Doorn. vtpm: virtualizing the

trusted platform module. In *USENIX-SS'06: Proceedings of the 15th conference on USENIX Security Symposium*, pages 21–21, Berkeley, CA, USA, 2006. USENIX Association.

[3] L. Chen, R. Landfermann, H. Löhr, M. Rohe, A.-R. Sadeghi, C. Stüble, and H. Görtz. A protocol for property-based attestation. In *STC '06: Proceedings of the first ACM workshop on Scalable trusted computing*, pages 7–16, New York, NY, USA, 2006. ACM Press.

[4] Y. Chevalier, L. Compagna, J. Cuellar, P. H. Drieslma, J. Mantovani, S. Mödersheim, and L. Vigneron. A high level protocol specification language for industrial security-sensitive protocols. In *Proceedings of Workshop on Specification and Automated Processing of Security Requirements (SAPS 2004)*, 2004.

[5] T. Dierks and E. Rescorla. The Transport Layer Security (TLS) Protocol Version 1.1. Technical report, IETF Network Working Group, 2006.

[6] D. Dolev and A. Yao. On the security of public key protocols. In *Proceedings of the IEEE 22nd Annual Symposium on Foundations of Computer Science*, pages 350–357, 1981.

[7] T. Garfinkel, B. Pfaff, J. Chow, M. Rosenblum, and D. Boneh. Terra: A virtual machine-based platform for trusted computing. In *SOSP '03: Proceedings of the nineteenth ACM symposium on Operating Systems Principles*, pages 193–206, New York, NY, USA, 2003. ACM Press.

[8] Y. Gasmi, A.-R. Sadeghi, P. Stewin, M. Unger, and N. Asokan. Beyond secure channels. In *STC '07: Proceedings of the 2007 ACM workshop on Scalable trusted computing*, pages 30–40, New York, NY, USA, 2007. ACM.

[9] K. Goldman, R. Perez, and R. Sailer. Linking remote attestation to secure tunnel endpoints. In *First ACM Workshop on Scalable Trusted Computing*, Fairfax, Virginia, November 2006.

[10] 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*, 2004 2004.

[11] C. Krauß, F. Stumpf, and C. Eckert. Detecting Node Compromise in Hybrid Wireless Sensor Networks Using Attestation Techniques. In *In Proceedings of the Fourth European Workshop on Security and Privacy in Ad hoc and Sensor Networks (ESAS 2007)*, *Lecture Notes in Computer Science*, Cambridge, UK, July 2007. Springer-Verlag.

[12] L. Lamport. Password authentication with insecure communication. In *Commun. ACM*, volume 24, pages 770–772, New York, NY, USA, 1981. ACM Press.

[13] J. Liedtke. On Micro-Kernel Construction. In *SOSP '95: Proceedings of the fifteenth ACM Symposium on Operating Systems Principles*, pages 237–250, New York, NY, USA, 1995. ACM Press.

[14] H. Löhr, H. V. Ramasamy, A.-R. Sadeghi, S. Schulz, M. Schunter, and C. Stüble. Enhancing grid security using trusted virtualization. In *Second Workshop on Advances in Trusted Computing (WATC'06)*, Tokyo, Japan, November 2006.

[15] A.-R. Sadeghi, M. Scheibel, C. Stüble, and M. Wolf. Play it once again, sam - enforcing stateful licenses on open platforms. In *2nd Workshop on Advances in Trusted Computing (WATC '06 Fall)*, Tokyo, Japan, November 2006.

[16] A.-R. Sadeghi, M. Selhorst, C. Stüble, and M. Winandy. TCG Inside? A Note on TPM Specification Compliance. In *Proceedings of the First ACM Workshop on Scalable Trusted Computing (STC'06).*, 2006.

[17] A.-R. Sadeghi and C. Stüble. Property-based attestation for computing platforms: caring about properties, not mechanisms. In *NSPW '04: Proceedings of the 2004 workshop on New security paradigms*, pages 67–77, New York, NY, USA, 2004. ACM Press.

[18] R. Sailer, T. Jaeger, X. Zhang, and L. van Doorn. Attestation-based policy enforcement for remote access. In *CCS '04: Proceedings of the 11th ACM conference on Computer and communications security*, pages 308–317, New York, NY, USA, 2004. ACM Press.

[19] E. Shi, A. Perrig, and L. V. Doorn. BIND: A Fine-Grained Attestation Service for Secure Distributed Systems. In *SP '05: Proceedings of the 2005 IEEE Symposium on Security and Privacy*, pages 154–168, Washington, DC, USA, 2005. IEEE Computer Society.

[20] F. Stumpf, M. Benz, M. Hermanowski, and C. Eckert. An Approach to a Trustworthy System Architecture using Virtualization. In *Proceedings of the 4th International Conference on Autonomic and Trusted Computing (ATC-2007)*, volume 4158 of *Lecture Notes in Computer Science*, Hong Kong, China, 2007. Springer-Verlag.

[21] F. Stumpf and C. Eckert. Enhancing Trusted Platform Modules with Hardware-Based Virtualization Techniques. In *Proceedings of the Second International Conference on Emerging Security Information, Systems and Technologies (SECURWARE 2008)*, Cap Esterel, France, August 25-31, 2008. IEEE Computer Society.

[22] F. Stumpf, O. Tafreschi, P. Röder, and C. Eckert. A Robust Integrity Reporting Protocol for Remote Attestation. In *Second Workshop on Advances in Trusted Computing (WATC'06 Fall)*, Tokyo, Japan, November 2006.

[23] Trusted Computing Group. TCG TPM Specification Version 1.2 Revision 103, Structure of the TPM. Technical report, TCG, 2007.

[24] Trusted Computing Group. Infrastructure Subject Key Attestation Evidence Extension Version 1.0, Revision 5. Technical report, TCG, 2005.

[25] Trusted Computing Group. TCG TPM Specification, Architecture Overview. Technical report, TCG, 2007.

[26] Trusted Computing Group. Trusted Platform Module (TPM) specifications. Technical report, TCG, 2008, `https://www.trustedcomputinggroup.org/specs/TPM`

[27] L. Viganò. Automated Security Protocol Analysis with the AVISPA Tool. In *Proceedings of the XXI Mathematical Foundations of Programming Semantics (MFPS'05)*, volume 155 of *ENTCS*, Elsevier, 2005.