ATTESTED GENUINENESS IN SERVICE ORIENTED ENVIRONMENTS

Anders Fongen, Federico Mancini
Norwegian Defence Research Establishment (FFI)
Norway
anders.fongen@ffi.no

Abstract

Authentication merely proves the identity of the parties during a SOA transaction. In order to trust the bona fide conduct of an operation there is also a need for trusting the software. The parties should provide a proof of genuineness *which can be validated in the other end. In this paper, two different approaches to hardware assisted software integrity control is presented. They both combine with an identity management service to provide* attested genuineness *for improved trust in the transaction. One approach employs the well known Trusted Platform Module hardware unit, the other is based on custom design hardware.*

Keywords

Integrity Protection, Trusted Platform Module, Identity Management

1. Introduction

In a service oriented (SOA) environment, the conduct of the transactions between a client and a server is regulated by a service contract which describes technical and legal aspects of the service. The contract could require, e.g., that human clients have particular qualifications or that the service is to be used only for a specific purpose.

The confidence in the appropriate conduct of a transaction relies on a chain of trust relations. Cryptographic keys may be used to bind the transport endpoint to a particular identity, while the procedures relating to the issuance and use of the keys may bind that identity to a certain entity (person) which was subject to an authorization decision. Chained together, these two separate relations justifies the trust that a verified digital signature ensures appropriate conduct.

Obviously, the strength of the security protocol and cryptographic operations are unimportant if the management of credentials is flawed. In the same manner, the establishment of identity is not necessary if the trust in bona fide conduct can be obtained through other mechanisms.[1]

For this purpose the concept of *genuineness* is introduced. The term indicates the ability and commitment of an actor to meet the requirements of a service contract. Since this property always includes non-technical properties (loyalty, competence, experience etc.), the virtue of genuineness will rely on trust chains spanning over technical and non-technical security arrangements.

The term genuineness may be a better property for the client-server relation than authenticity, because it describes more directly the risk for abuse, fraud or other misconduct during the transaction. An authenticated transaction is, e.g., not able to guard against malware which modifies the intended behavior during the operation.

How can genuineness be demonstrated or proven? The mechanism suggested in this paper is by *attestation*. This means that a trusted third party enters into the negotiations between the client and the server and provides attestation for the virtues of the parties, in the form of cryptographically sealed documents. The attestation may be used in the subsequent client-server protocol interactions for verification by the other part.[2]

1.1. Service Integrity

The intended behavior of an actor is dependent upon the integrity of the software being executed on its behalf. Thus, in order to establish the genuineness of a transaction, it is necessary to assess the present integrity of the software system, as well as the potential loss of integrity at a later instant.

There are two prevalent approaches to support service integrity in a run-time platform: One is through *separation*, where the operating system enforces rigid isolation between activities to avoid unwanted influence (like malware). The other is by *inspection*, through which the run-time system is able to detect if the program code has been modified from its "good" state. These two approaches may well be combined.

1.2. Identity Management

The term *identity management* (IdM) refers to a set of services and procedures for the management of identity information. In the perspective of this paper IdM offers management of key data (private/public keys) and attributes (for access control decision etc.), and may issue *credentials* which attest the relation between the identity (e.g., a unique name) on one hand, and public keys and attributes on the other.

The credentials bind this information together with a digital signature of the *identity provider* (IdP), made with a key which is well known and trusted. The *X.509 public key certificate* [3] is a well known example of an IdM credential.

Through the centralized management of identity information in a separate architecture with the potential for an efficient distribution mechanism for credentials, the IdM represents a scalable and manageable approach to the key distribution problem. The complexity of key distribution is otherwise often overlooked in the design of cryptographic protocols. [4]

The contribution of this paper is a set of proposed protocols which offer attested genuineness, in particular targeted for

lightweight nodes which are found in sensor networks and in the expected ”Internet of Things”.

The remainder of the paper is organized as follows: In Section 2 the concept of integrity protection will be justified and two different approaches will be presented. In Section 3 the prototypical IdM used for the experiments related to these efforts will be presented. The discussion of how to employ the TPM unit in combination with identity management is given in Section 4. The paper will present the conceptual design of a lightweight integrity protection unit in Section 5. Related research is presented in Section 6, and the paper concludes and suggests further research in Section 7.

2. Software integrity protection

Perceived genuineness relies on the trust between the actors of a transaction, on the confidence that both operators and software are operating in accordance with the service agreement. As mentioned in the introduction, the trust in the operator personnel relies on a chain from the cryptographic operations in the protocol to the credential issuing policy.

The trust in the software relies on the correctness of the programming, maintenance, version control, deployment, malware protection and the isolation between activities in the execution platform, just to mention a few. The security properties of the software may be assessed and attested by a trusted third party. The assessment will include an audit of the development process as well as run-time inspection of code.

2.1. MLS and MILS systems

In classic security literature, the separation of activities inside a common execution platform is essential to preserve the confidentiality and integrity of data.[5] Controlled communication between activities is required in order to provide the necessary coordination between otherwise isolated activities.[6] In Multi Level Security systems (MLS) , data on different confidentiality levels is processed in the same system and a flow of data from “low” to “high” confidentiality is sometimes permitted.

The protection of software code from tampering is essential to preserve the integrity of operations, but MLS systems does not offer any mechanisms by which the owner of the system can present a *proof* of software integrity. MLS systems may be subject to evaluation, e.g., according to Common Criteria [7] and obtain an EAL level as a statement of the general robustness of an operating system. The EAL doesn’t state anything about the actual integrity state of the running software and whether is has been tampered with lately.

MLS systems has been found to be extremely complex and costly, and simpler models like MILS (Multiple Independent Levels of Security) have been devised where the isolation property is the focus of the design, more than the communication between activities.[8] A MILS kernel can be made much smaller than an MLS kernel, resulting in lower cost of development and evaluation.

2.2. Trusted Platform Module

A software inspection mechanism which in itself is implemented in software is also vulnerable to the same integrity problems. Therefore, hardware units should be designed to assist the inspection of software integrity and related cryptographic operations.

The Trusted Platform Module (TPM) has been designed for this purpose.[9] It is a crypto processor with several mechanisms related to asymmetric key generation and storage, cryptographic operations, HMAC (Hashed Message Authentication Codes) calculation, etc.

The proliferation of the TPM (it is found in most PCs sold today, and in a growing fraction of tablet computers) makes it interesting for investigation: Does it have the necessary properties to support the desired mechanisms for attestation of software integrity?

An overview of its functions relevant to this work will be given in Section 4. For further technical details, please refer to the TGC (Trusted Computing Group) documents.[9]

2.3. Lightweight hardware protection units

We also present the design of a hardware unit even simpler than the TPM. The unit is able to inspect code memory and calculate the hash values itself, and include in the computation keys which are kept inside the unit (and never used for any other purpose). The value from this computation is never transported outside the computer node, but used as key in subsequent symmetric cryptographic operations.

The correct hash values are not known to the protection unit, but only to the IdP (identity provider), which issues the keying material to the client accordingly. The server can only decrypt ciphertext from the client if the protection unit has calculated the correct hash values.

The details of the unit will be formally presented in Section 5, where it will be shown that the unit is highly robust against physical attacks, provided that an identity management system is in place.

The claim of the discussion in this section is therefore that *the presence of an IdM offers potential hardware solutions to software integrity that is stronger than a standalone TPM chip.*

3. The GISMO IdM

The presence of an identity management system is essential to the efforts related to attested genuineness. The IdM will serve as a trusted third party and issue security statements which are protected with its digital signature.

For the efforts on establishing a proof-of-concept prototype for attested genuinity, an existing IdM called “GISMO IdM” has been used. GISMO IdM was developed to study the necessary properties for an IdM used in a multi-domain wireless mobile network used by a coalition tactical force.[10]

In GISMO IdM, existing PKIs are kept for reasons of investment protection, but encapsulated by a number of Identity

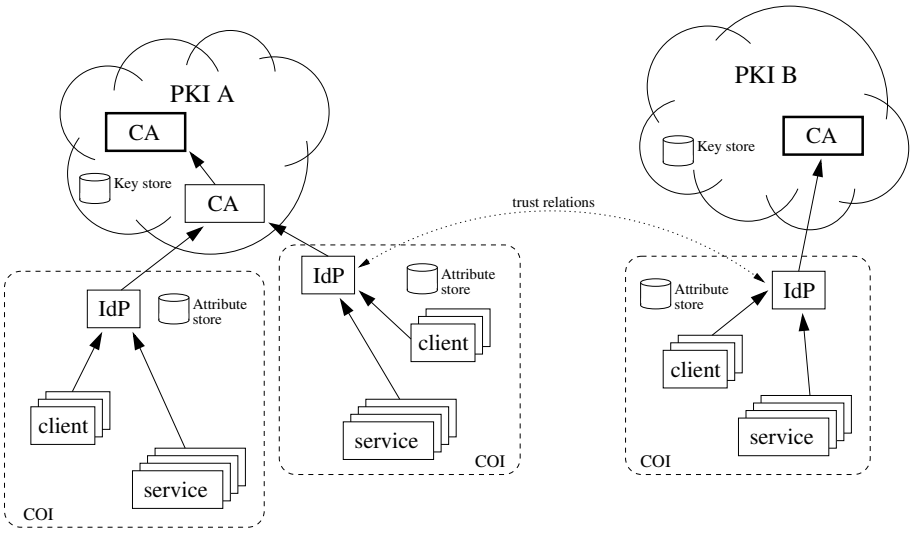

Fig. 1. The functional components of a federated IdM. Observe that the IdP serves one single COI, and the trust relations are formed between COIs, not domains. Key management is handled by the PKI whereas the attribute management is done by the IdPs on the COI level

Subject Distinguished Name
Subject Public Key
Subject Attributes
Valid from−to
Issuer Distinguished Name
Issuer's Signature

Fig. 2. The structure of the Identity Statement

Providers (IdP), each serving a "community of interest" (COI). The members of a COI share the IdP's public key as their trust anchor. The IdP issues *Identity Statements* (IS) to attest the public key and attributes of a subject. Information about the software integrity status in a computer may be stored among the attributes. The IS is given a short lifetime and sealed with the signature of the IdP. Due to the short lifetime, no revocation arrangement is necessary.

The architectural overview of GISMO IdM is shown in Figure 1. The key properties of are explained in the following paragraphs:

3.1. Authentication support

The identity provider (IdP) issues *identity statements* which bind the public key of a subject to its identity, much like X.509 Certificates. Identity Statements (IS) are issued to local subjects known to the IdP, as well as to subjects who can display an IS issued by a different IdP to which this IdP has a *trust relationship*. The structure of an IS is shown in Figure 2.

The subjects (either client or server) authenticates themselves during the service invocation by the use of their identity statements and their private keys. Different protocols have been designed with the purpose of generating as little network traffic and as few protocol round trips as possible. One of these protocols is shown in Figure 4.

3.2. Integrated access control

Included in the identity statement is a set of attributes which describes features of the subject in the form of name-value pairs. The attributes can describe *roles* of the subject and enter into access control decisions based on the Role Based Access Control (RBAC) or Attribute Based Access Control (ABAC) model. They can also describe other properties of the subject, e.g., preferred language, proficiency level etc.

The attributes are sealed inside the identity statement with the signature of the IdP, so they cannot be changed once issued.

3.3. Cross-domain operations

Clients can invoke servers in a different COI as indicated in Figure 1, provided that there exists a trust relationships between the two COIs. A client obtains an identity statement from its IdP, then passes on that IS to the IdP of a foreign COI. The foreign IdP can issue a *guest IS* containing the same information, but signed by the foreign IdP. Since the guest IS's signature will be trusted by servers in the foreign domain, it can be used to authenticate to these servers. Server authentication requires a *cross domain IS* issued from one IdP to the other, so a signature chain back to the client's trust anchor can be constructed, which is taken care of in the protocols as shown in Figure 3.

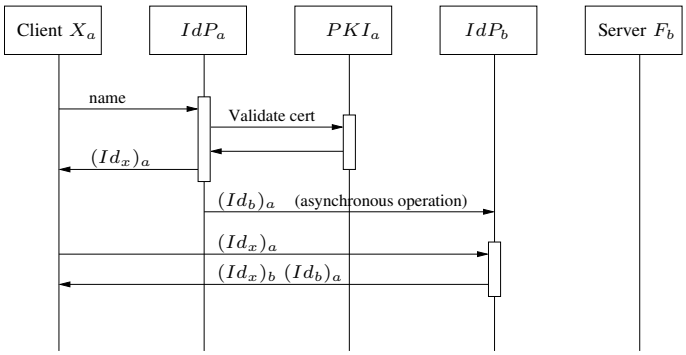

Fig. 3. The identity statement issuing protocol. The IdP of COI A, termed IdP_a, issues a "native" identity statement to the client, which is given to IdP_b, which in turn issues a guest identity statement. The term PKI_a denotes a set of certificate validation services in COI a.

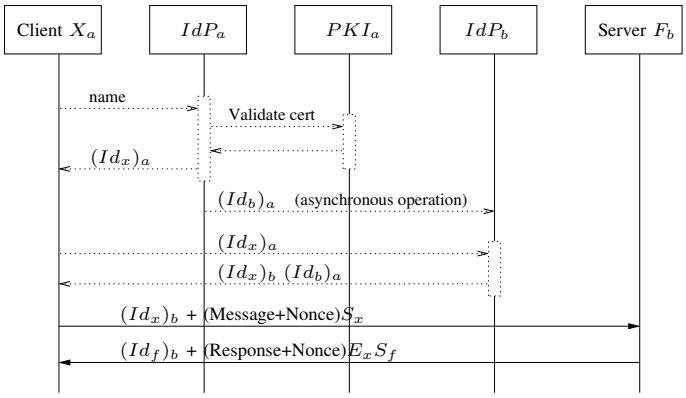

Fig. 4. The authentication protocol for a stateless service. Requests are not replay protected since replay is not considered as a threat, but the response need to be protected for reasons of response replay and information compromise. For the sake of integrity protection, the request is signed. The encryption of the response is a part of the authentication scheme, not a privacy measure.

3.4. Identity Management used for attested genuineness

One contribution of this paper is to show how identity management can support integrity control through the use of identity statements. An identity statement can contain information on how the software of the computer may be trusted, which means that the IS, which is issued on behalf of a subject (e.g., a person) now contains information about the computer through which the subject is operating. Consequently, the identity statement must be bound both to the computer and the subject.

The binding of computer and subject requires that keys belonging to both computer and subject is employed in the IS issuing process. Keys that belongs to the computer must be stored in tamperproof hardware so it cannot be moved, and the subject's keys must be safeguarded from misuse by others. The binding must be verified when the IS is issued, but during the authentication phase it is sufficient to demonstrate the computer's key. That choice was made in order to achieve compatibility with the default authentication process (where only one key is in use), and because the computer's keys are expected to be reserved by a single activity and kept out of reach from other processes.

It is a requirement that a stolen or lost computer should be expelled from the COI and not be given any IS. We can achieve that through the same IdM procedures as with subjects: They are removed from the IdP's centralized registry and will not any longer be recognized for attestation of integrity protection. However, computers without hardware keys and integrity control will still be able to operate on behalf of their users.

4. TPM based integrity protection

The TPM chip enhances the security of most systems through the possibility to securely generate, use and store cryptographic keys and other sensitive data in a shielded hardware component. This alleviates the problem of private key protection, but also allows for more advanced services, like *attestation* and *sealing*. Attestation means that it is possible to demonstrate to a third party that some key or data was generated and kept inside the TPM in a tamperproof manner. The data could be the list of software running on the machine since its boot. In other words, one could prove that the machine is in a *trusted state*, since it is running only genuine and trustworthy software.

4.1. TPM Based Attestation

A platform configuration can be represented by the list of applications that have been loaded in memory and executed since the machine was booted. If the list is complete and correct then it is possible to detect unauthorized applications loaded at some point, rendering the platform potentially compromised. The *remote attestation* process, consists in two phases: measuring the platform integrity and reporting it to a third party.

4.1.1. Integrity Measurement. When using the TPM, one can *measure* the application code while it is loaded into memory and store its hash in a TPM register called PCRs (Platform Configuration Registry). This process relies on the concept called *Root of Trust for Measurement* (RTM). Some trustworthy entity starts measuring the first piece of code (the BIOS) loaded into memory and stores this measurement in the TPM. This piece of code is usually called *Core Root of Trust for Measurement* (CRTM) and resides in a read-only portion of the BIOS itself. If we can trust this first measurement to be genuine then we can build a chain of transitive trust relations by passing control to the next code image and repeat the process until the OS is loaded. At this point the PCRs will contain an incremental hash representing the boot sequence

of the machine that we can trust both because of the RTM, and because the PCR values only can be *extended*, but never overwritten [11].

All the measurements in the TPM can now be retrieved and the whole boot sequence reconstructed. The measurement process will detect malware inserted into the sequence as long as the CRTM itself was not compromised. In this case the CRTM is also called a Static CRTM (S-CRTM) because the trust chain can be built only at boot time and cannot be restored without restarting the machine and repeat all measurements.

Drawbacks of this approach are that we must trust the BIOS, where the CRTM is usually implemented. The chain of trust stops when the OS takes control of the platform. Trust can be established only once at boot time. Given an attacker with physical access to the machine, the BIOS might be re-flashed or the PCR values reset [12] and arbitrary measurements could be loaded into the TPM. Besides, the same method cannot defend against a bug in the OS which could be exploited to store false measurement about the software running on the platform after the OS was put in control.

To solve these problems a new type of RTM was introduced called Dynamic CRTM (D-CRTM) [13]. The trust in the platform is now established by executing, at any time, a special instruction on the processor that initializes a special PCR value in the TPM and creates a sanitized environment to run some given code safely. The main difference from a S-CRTM is that this special PCR can be extended only through a special processor cycle and cannot be simulated by software. The code can be executed safely even if the running OS was compromised, and its execution measured securely and stored in the TPM.

Once the OS takes control, it becomes almost impossible to guarantee the integrity of every piece of software running on the machine. The reason is that OSes should play the part of the RTM, but they can rarely be considered secure enough for that role. Every application could be run separately by using, for instance, the D-CRTM, but this would be unrealistic for ordinary personal computers.

4.1.2. Remote Attestation. Once the integrity measurements are securely stored in the TPM PCRs we need to be able to take them out of the TPM and report them securely to a trusted third party. These values are signed by the TPM using a special cryptographic key, which is guaranteed to never have been left unprotected outside the TPM and to sign only data generated inside the TPM. The third party can verify this signature and make sure that the PCR values were neither tampered with nor forged.

4.2. TPM Key Pairs

The TPM contains several key pairs which is used in cryptographic operations. The *Endorsement key* is created during manufacturing and is never used to sign data, but only to prove to a third party that a genuine TPM is involved in a transaction. This is achieved by using the corresponding key certificate, called *Endorsement certificate* which is (indirectly) signed by Verisign Corp. The endorsement key is denoted EK and the corresponding certificate $Cert_{ek}$.

Out of privacy concerns, rather than using directly the EK in actual transactions, the TPM can create another type of key to prove its genuineness without compromising its identity. These are called *Attestation Identity keys*, denoted AIK. These keys are mentioned previously used to sign PCR values and new created keys in place of EK. However, because the EK cannot be used to sign anything, a trusted third party must issue a certificate stating that an AIK was indeed created by a real TPM. This external certificate authority is called *Privacy CA* (PCA).

The PCA can certify AIK due to the Endorsement Certificate $Cert_{ek}$ which is included as a part of the certification request. The generated AIK certificate, $Cert_{aik}$, is sent back to the TPM encrypted with EK, so that only the real TPM can decrypt it and use it.

Legacy keys, termed LK, can be created at any time and are the only ones that can be used for both signing and encryption. The private part of the key never leaves the TPM, and signature and decryption operations take place inside the unit. Legacy keys can be attested by the AIK as explained in Section 4.3.2.

There are also other type of keys in the TPM, but they are not relevant to the work presented in this paper. In the following discussion, signatures made with the AIK are denoted $Sign_{aik}$ and signatures made with LK are denoted $Sign_{lk}$.

4.3. Protocols of IdM operations

In this section we show how it is possible to integrate the TPM attestation capabilities in the IdM protocol illustrated in Section 3, and achieve genuineness protection. The protocols for two different IdM operations related to the use of TPM protection will be presented:

1) During the initiation of a computer the keying material in the TPM has to be approved by the IdM in a fashion similar to that of a PCA, so that identity statements later can be issued regarding the public key of that TPM. This arrangement also allows us to revoke the key from the IdP in order to expel the computer from the community.
2) Prior to a client-server transaction an identity statement must be issued that attests a newly generated legacy key and binds that key to the subject which operates the computer. The key being used in this operation is attested by the AIK.

The following discussion assumes the existence of the crypto operations $C = E(K, P)$ (encrypt plaintext P with key K and produce ciphertext C) and $P = D(K, C)$ (decrypt C into P with key K).

4.3.1. TPM approval. As explained in Section 4.2, the TPM contains an *endorsement key* EK which is certified by Verisign Corp.

During initiation of the computer the attestation identity key AIK will be generated and a self-signed *certificate request* for the public part of AIK (denoted CR_{aik}) sent to the IdP. Together with the certificate request, $Cert_{ek}$ will also be sent to the IdP. $Cert_{ek}$ can be validated since we have trust in the issuing certificate from Verisign Corp.

The certificate request does not prove that this is a TPM-based public key, and it does not indicate which TPM it was created in. In order to bind the resulting $Cert_{aik}$ to the TPM it is encrypted with EK and returned to the TPM. This response message is written $E(EK, Cert_{aik})$. Observe that the IdP now serves the same function as the *privacy CA* described in Section 4.2.

Only the TPM that claims to have issued the AIK will be able to decrypt that response and make use of the AIK certificate. Subsequent use of the AIK should therefore present a proof that the certificate has been decrypted in order to demonstrate that it is contained in a TPM. The proof can be the certificate itself ($Cert_{aik}$) or a certificate digest.

4.3.2. Identity Statement Issue. An identity statement for a subject operating on a TPM-equipped computer should bind the two entities together using the subject's id and the keying material of the TPM. The key (with the private part resident inside the TPM) will later be used to sign and decrypt protocol messages. AIK cannot sign anything outside the TPM, so a new legacy key for that purpose must be created (denoted LK). The legacy key need to be attested by the AIK in order to be approved by the IdP.

The TPM attests a legacy key through the following data structure: $(SNo_{aik}, (Dig_{lk})Sign_{aik})Sign_{lk}$, i.e., the digest of LK signed by AIK concatenated with the serial number of the $Cert_{aik}$ and signed by LK. Sent to the IdP together with LK the IdP can validate that LK is attested by AIK and trust that the key is generated by the TPM. $Cert_{aik}$ is already in the IdP so it needs not be sent. This also solves the certificate verification problems mentioned in [14].

In addition, the subject's identity/name (Nm_s) must be sent and demonstrated through a digital signature made with the subject's private key ($Sign_s$). For replay protection, the signature must also cover the structure above. The entire data structure to be sent for IS issue is:

$$(Nm_s, LK, (SNo_{aik}, (Dig_{lk})Sign_{aik})Sign_{lk})Sign_s \quad (1)$$

and the response from the IdP is an identity statement on this form

$$(Nm_s, LK, Attr, ValPer, Nm_{idp})Sign_{idp} \quad (2)$$

Figure 5 shows the TPM approval and IS issuing protocol as an interaction diagram.

The structure of the identity statement is identical to what is found in other applications of the GISMO IdM, i.e., it binds together a name (Nm_s), some attributes *(Attr)*, a validity period *(ValPer)* and a public key *(LK)*, and is signed by the trust anchor ($Sign_{idp}$). Consequently, the following interactions between a client and a server will use the unmodified

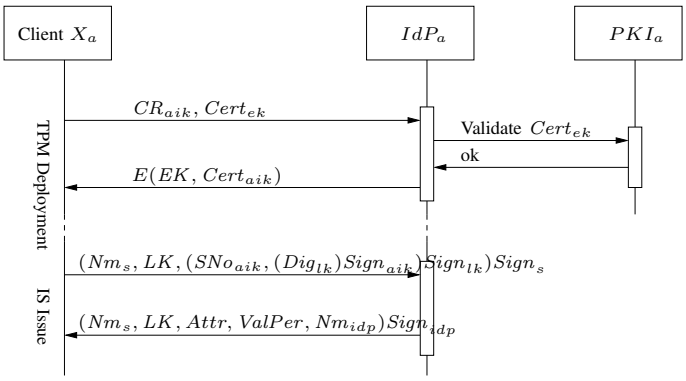

Fig. 5. The TPM deployment protocol, where the genuinity of the TPM is assessed, and the IS issue protocol, where the TPM key is bound to the subject which operates the computer.

GISMO protocols for authentication and invocation, e.g., as shown in Figure 4.

Using unmodified protocols for authentication and invocation allows TPM-equipped computers to inter-operate with other computers not using the TPM for attestation. The attributes of the IS will indicate the difference if the application wish to distinguish between the two. Interoperability is also the reason for the choice of legacy keys, although their use is not encouraged. They allow a TPM-equipped node to perform encryption and signature operations with the same key pair. This is a requirement by the protocol shown in Figure 4.

5. Integrity Protection Unit

The TPM chip itself is indeed quite secure. Even though it is not mandated to be tamperproof against physical attack it is extremely difficult to extract the private keys stored inside it or break the encryption of the data stored outside the TPM. However, if the data flow between the TPM and the application using it could be intercepted, passwords could be compromised and false measurements could be fed to the TPM [15]. This could be critical in particular if the device is exposed on the field, like sensor nodes are. Such devices have also limited resources, so a TPM might be considered too expensive in terms of power and costs. The unprotected nature of the devices requires that the TPM and the roots of trust should be made physically tamperproof and no sensitive information should be easily accessible (i.e., stored unencrypted). The device would be compromised if the CRTM code is replaced by malicious code.

For a simple embedded device like a sensor node, the TPM probably not the best choice for integrity protection. A simpler chip, integrated with the main board of the node and which only provides basic hashing, symmetric crypto operation and storage functions will suffice. Lowering the complexity also reduces the attack surface.

The paper will now commence with a presentation of a suggested design of a hardware unit which can serve the

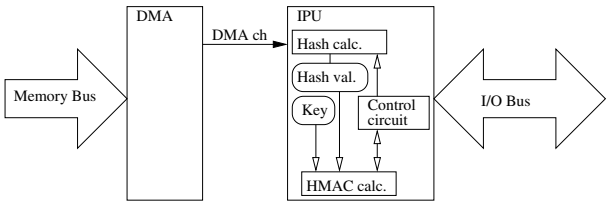

Fig. 6. The structure of the Integrity Protection Unit

integrity protection mechanism according to the requirements set forth in the discussion above. The design is targeted on sensor nodes or other energy efficient embedded units without virtual memory. It offers encapsulation of secret keys and calculation of memory hash values and combines these operations so that the unit is protected even against realistic physical attacks. Since the unit is expected to operate in battery powered devices only symmetric crypto algorithms is being used (asymmetric crypto requires more energy per operation).

The unit is called *Integrity Protection Unit* (IPU) and a structural overview is shown in Figure 6

The protocol of the unit consists of a simple challenge-response mechanism where the challenge enters into an HMAC calculation together with the calculated memory hash values and the secret key. The resulting HMAC value is used as a symmetric crypto key for subsequent communication.

The hash value calculation is invoked through the reset vector of the processor. The memory portion to hash is brought directly to IPU by a connected Direct Memory Access (DMA), so that an attacker cannot forge a fake hash and feed it to the unit as it can happen with the TPM. This requires also that the amount of data to be hashed does not exceed a certain threshold, or the computational load on the IPU would become too great.

A HMAC function f of the unit can be defined as follows:

$$hmac = f(K, h(mem), challenge) \tag{3}$$

The hash function h is not evaluated for each call to f, but during bootstrap and later when felt necessary. The result is cached for use as a parameter in the f function. It is imperative that the secret key K is never employed in any other operation than f, where $h(mem)$ always is a parameter. The parameter $challenge$ is supplied by the client.

HMAC-based authentication does not secure the subsequent message traffic from a man-in-the-middle attack, but the HMAC function may be used to establish a shared secret between the client and the service which can be used for message encryption in order to solve that problem. The next section will describe a protocol through which a trusted third party (an Identity Provider) can manage keys and identity information and cooperate with the hardware unit to attest the genuineness of the service.

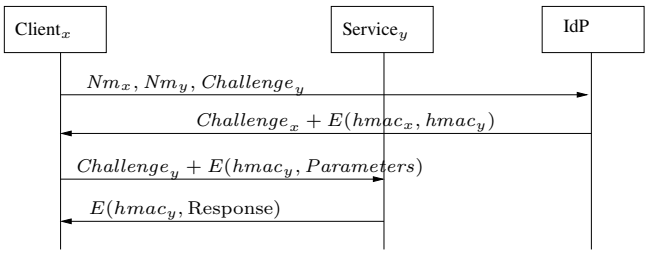

Fig. 7. The protocol for protection of genuineness, based on symmetric crypto keys and the HMAC function given in Equation 3.

5.1. IPU protocol for attestation of genuineness

It will now be shown how a tamper proof HMAC function as described in Section 5 can support a protocol which allows a client and a server to trust the genuineness of the other part. Keep in mind that both actors (client and server) are integrity protected with the hardware unit as described earlier in this section.

The protocol has these properties:

- Only the identity provider (IdP) knows the secret keys and the valid memory hash values of the service and the client.
- No online connections between the service and the IdP is necessary. Only the client needs to invoke the IdP.
- Genuineness control of the client happens during the invocation of the IdP.
- Genuineness control of the service and the service invocation happen in the same protocol round trip in order to save network resources.
- Client and service need software libraries for symmetric crypto operations.

Figure 7 shows the elements of this protocol. It assumes that both the client and the server contain an IPU with their secret key, memory hash and HMAC function (as shown in Eq.3) inside. It also assumes that the IdP knows the secret keys and "approved" memory hash values of the two actors and can calculate the same $hmac$ value as them.

Furthermore, it assumes the existence of the symmetric crypto operations $C = E(K, P)$ (encrypt plaintext P with key K and produce ciphertext C) and $P = D(K, C)$ (decrypt C into P with key K). In the figure, $hmac_x$ denotes the result from Equation 3 using the client's key, the approved or calculated memory hash value and $Challenge_x$. $hmac_y$ is to be interpreted similarly for the server.

Note that neither $hmac_x$ nor $hmac_y$ values are sent through the network in plaintext, but calculated inside computers and used as a key for subsequent message transport. The client needs the $hmac_y$ value which is sent from the IdP encrypted with $hmac_x$, $E(hmac_x, hmac_y)$. In order for the client to communicate with the server it need to calculate its own correct $hmac_x$ value. The server will not be able to understand the client request unless it has calculated $hmac_y$ correctly.

5.2. Authentication

Since the presence of secret keys and correct memory hash value must be demonstrated during the protocol transaction, confidentiality, authenticity and genuineness is assured. The protocol requires the client to identify itself and the desired server (Nm_x, Nm_y), and the use of a false name will result in a response useless to the client.

Due to the protocol design, the client's identity remains with the IdP and will not be known to the server. This might be considered as enhanced privacy, but it also hinders the server from making local access control decisions.

5.3. Access control decisions

Prior to the issuing of the $E(hmac_x, hmac_y)$ value, the IdP may make *access control decisions* based on their identities. If the transaction between the two should not be allowed, the IdP will simply reject to issue the encrypted response.

One unsolved problem is the unlimited validity time of the HMAC values. The client can use the same HMAC value repeatedly and in any time instant in the future. The server nodes may have limited resources so a "memory" that can detect duplicate challenges is not an appealing requirement. In a mobile environment however, this effect could even be an advantage since the client could "tank up" its memory with HMAC values that could be used to communicate with servers even when there is no communication path to the IdP.

The protocols described in this section related to management of IPU-equipped computers and the use of an IdP are not closely related to the design of the GISMO IdM, which uses asymmetric crypto for its services. The protocols related to the IPU uses only symmetric crypto, for reasons of energy efficiency. Our proposed IPU protocol design will require some straightforward additions to the IdP software.

5.4. Certification of correctness and robustness

The separation of duties found in the relation between the server and the IdP, i.e., the computer calculates the memory hash value blindly and the IdP tests it for correctness, facilitates external auditing of software. A well regarded company can evaluate the application and the system software in the computer, as well as the software development methods used, in order to assess the correctness and robustness of the software.

6. Related research

Research concerning the use of tamperproof hardware units to strengthen the trust in different kind of information systems has been going on for quite some time. One of the main goals in the design of the TPM unit has been to be able to attest the integrity of a platform so that a third party could decide whether to trust it or not. This has been partly achieved by defining Static and Dynamic Root of Trusts as explained in Section 4.1 and through remote attestation protocols [16], [17], but these approaches are only able to measure the integrity of the system up to the point when the operative system is loaded. It is considered difficult to guarantee the integrity of the platform at a later stage.

Besides, none of the infrastructure elements needed to perform remote attestation has been commercially implemented yet, so it is not clear how effective it might actually be once deployed on a large scale. The only notable exceptions of TPM based commercial applications have been BitLocker [18] and HP ProtectTools [19], which are mostly used to protect data stored locally on the machine. Most recently, TPM assisted local integrity verification based on Trusted Boot and hardware firmware verification has been employed to securely boot the new Microsoft and Google operative systems.[20], [21] No unanimously accepted solution for remote attestation or run-time integrity protection exists yet, although various approaches have been proposed [22], [23], [24], [25] and research in this area is still active.

The idea of using a TPM to improve Identity Management Systems is not completely new. The use of Trusted Computing (TC) concepts with Single Sign On (SSO) systems has been investigated in [26], [27], while the work in [28] about a TPM based protocol for OpenID is the one that comes closest to the ideas presented in Section 4.3. The reason for the similarity between the two protocols is the fact that both use TPM identity keys (AIKs) as a base to establish trust in the identity of the platform and to certify asymmetric keys and integrity measurements. So both works bind and delegate the identity of the user to that of the TPM.

However, there are some important differences as well: One is that the intended operational environments for OpenID and GISMO IdM are not the same. While OpenID is mostly intended to simplify web services authentication, GISMO IdM is designed to provide both authentication and access control in a tactical environment with relatively low resources and a well defined community of interest. As a consequence, the underlying communication protocol are essentially different. Another difference is that one of the motivation for the use of a TPM in OpenID[28] was to improve user experience on the web by eliminating the need for passwords in the authentication process. Passwords are in fact substituted by TPM-protected user certificates. However, the OpenID IdP does not have any guarantee that such certificate was adequately protected on the local machine. In the GISMO IdM protocol proposed in this paper the TPM keys are cryptographically bound to a user identity, so that we can achieve a higher degree of integrity assurance. In fact, both the platform and the user must have been subject to some verification process before the identity statements are issued. Finally, while in OpenID the TPM involvement is limited to the authentication process which involves the IdP, we extend the use of the TPM certified key to the transactions with the target service, hence improving the security of the system as a whole.

Several attempts to solve the problem regarding the use of TC and remote attestation with Wireless Sensor Nodes

(WSN) have been observed. Some support and optimize the use of the TPM for WSNs, either by assuming that each sensor node is equipped with a TPM [29] or that only some selected nodes in a cluster have enough resources to support it [30]. Other consider the TPM too expensive in terms of hardware and power costs and propose alternative solutions, like the use of an ARM processor with Trustzone features [31] or software attestation [32]. This last protocol for software remote attestation, called SCUBA, is the one that is closest to the solution proposed in Section 5.

SCUBA is also based on the checksum of the code to be executed, but in order to reveal whether an attacker tries to forge the checksum the response delay from the node is measured. The assumption is that the time an attacker needs in order to create a fake checksum can be detected and therefore it is possible to invalidate the session in question. The reason why we do not need to time the response is that we can read the code and calculate its checksum when it is already loaded in memory for execution. In addition, an attacker might fake the checksum, but not the HMAC, since the key is protected inside a tamper proof hardware unit. An IPU with a preinstalled secret key would solve the symmetric key distribution problem of WSNs as long as an IdP is available during the startup of the nodes.

No related research on hardware units alternative to the TPM for use with WSNs has been found, nor on the use of IdMs to protect communication and integrity in WSNs. The approaches presented in this paper is therefore likely to be new.

7. Conclusion and future research

The purpose of this paper has been to show how the combination of an identity management service and hardware assisted integrity protection can offer higher assurance in a software system. The trust in the correct conduct of a transaction relies on a series of factors, and not all factors will be covered solely through an authentication process.

Two different approaches to hardware assisted integrity protection have been investigated. One was based on symmetric crypto, hash value calculations and direct memory access. This unit has not been prototyped, and is meant for simple embedded devices without virtual memory, e.g., sensor nodes. The hardware unit would need to be integrated into the motherboard of the computer and require a tailored design.

The other approach was designed around commodity hardware, the Trusted Platform Module, which supports a wide variety of configurations and are already installed in most personal computers. The TPM offers trusted bootstrap of the operating system, encapsulation of private keys and crypto operations. It is a rather complicated unit, and the software libraries for its API were incomplete and experimental. Parts of the TPM protocols were discovered through the reading of source code rather from the specifications.

Both approaches were possible to interface to our GISMO IdM, which issues the necessary credentials for authentication and integrity protection. The two hardware based protection mechanisms demanded completely different protocols, but the TPM based authentication protocols was shown to be compatible with ”non-TPM” protocols. Consequently, a TPM equipped node and a non-TPM equipped node are able to interoperate.

The integrity control offered by the TPM unit requires that the entire bootstrap process (from BIOS to OS) is able to employ the TPM mechanisms. A few operating systems do this, the most notable being recent versions of Microsoft Windows[20]. However, it is not established how that protection could be extended to cover also application code.

Future efforts on the matter of integrity protection will focus on the use of the TPM unit, and a laboratory is under development where the protocols and protection mechanisms can be tested under realistic threat conditions.

References

[1] A. M. Hegland, E. Winjum, and O.-E. Hedenstad, “A framework for authentication in nbd tactical ad hoc networks,” *IEEE Communications Magazine*, vol. 49, no. 10, pp. 64–71, 2011.

[2] A. Fongen, “Identity management and integrity protection in the internet of things,” in *3rd Int.Conf. on Emerging Security Technologies (EST2012)*, Lisbon, Portugal, September 2012, pp. 111–114.

[3] R. Housley, W. Polk, W. Ford, and D. Solo, *Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile*, RFC Editor, United States, 2002.

[4] A. J. Slagell, R. Bonilla, and W. Yurcik, “A survey of PKI components and scalability issues,” in *Proceedings of the 25th IEEE International Performance Computing and Communications Conference, IPCCC*, Phoenix, AZ, 2006, pp. 475–484.

[5] P. A. Loscocco, S. D. Smalley, P. A. Muckelbauer, R. C. Taylor, S. J. Turner, and J. F. Farrell, “The inevitability of failure: The flawed assumption of security in modern computing environments,” in *In Proceedings of the 21st National Information Systems Security Conference*, 1998, pp. 303–314.

[6] R. Spencer, S. Smalley, P. Loscocco, P. L. (national Security Agency, M. Hibler, J. Lepreau, and D. Andersen, “The flask security architecture: System support for diverse security policies,” in *in Proceedings of The Eighth USENIX Security Symposium*, 1998, pp. 123–139.

[7] *Common Criteria for Information Technology Security Evaluation*, ISO/IEC 15408, Sept 2012.

[8] J. Alves-foss, W. S. Harrison, P. Oman, and C. Taylor, “The MILS architecture for high-assurance embedded systems,” *International Journal of Embedded Systems*, vol. 2, pp. 239–247, 2006.

[9] Trusted Computing Group, “Trusted Computing Group,” http://www.trustedcomputinggroup.org/, online, Accessed Mars 2012.

[10] A. Fongen, “Architecture patterns for a ubiquitous identity management system,” in *ICONS 2011*. Saint Maartens: IARIA, Jan. 2011, pp. 66–71.

[11] Trusted Computing Group, “TCG PC Client Specific Implementation Specification for Conventional BIOS,” http://www.trustedcomputinggroup.org/, online, Accessed Mars 2012.

[12] E. R. Sparks, “A Security Assessment of Trusted Platform Modules,” Dartmouth College, Computer Science, Tech. Rep. TR2007-597, Jun. 2007. [Online]. Available: http://www.cs.dartmouth.edu/reports/TR2007-597.ps.Z

[13] D. Grawrock, *The Intel Safer Computing Initiative: Building Blocks for Trusted Computing*, ser. Engineer to Engineer Series. Intel Press, 2006.

[14] C. Latze and U. Ultes-Nitsche, “A proof-of-concept implementation of eap-tls with tpm support,” in *ISSA*, H. S. Venter, M. M. Eloff, J. H. P. Eloff, and L. Labuschagne, Eds. ISSA, Pretoria, South Africa, 2008, pp. 1–12.

[15] L. Chen and M. Ryan, “Attack, solution and verification for shared authorisation data in tcg tpm,” in *Proceedings of the 6th international conference on Formal Aspects in Security and Trust*, ser. FAST’09. Springer-Verlag, 2010, pp. 201–216.

[16] E. Brickell, J. Camenisch, and L. Chen, "Direct anonymous attestation," in *Proceedings of the 11th ACM conference on Computer and communications security*, ser. CCS '04. ACM, 2004, pp. 132–145.
[17] M. Pirker, R. Toegl, D. M. Hein, and P. Danner, "A privacyca for anonymity and trust," in *TRUST*, ser. Lecture Notes in Computer Science, L. Chen, C. J. Mitchell, and A. Martin, Eds., vol. 5471. Springer, 2009, pp. 101–119.
[18] Microsoft, "BitLocker," http://technet.microsoft.com/, online, Accessed Mars 2012.
[19] HP, "Embedded security for hp protecttools," http://h20331.www2.hp.com/Hpsub/cache/292199-0-0-225-121.html, online, Accessed Mars 2012.
[20] Microsoft, "Secured Boot and Measured Boot: Hardening Early Boot Components Against Malware," http://msdn.microsoft.com/en-us/library/windows/hardware/br259097.aspx, online, Accessed December 2012.
[21] Google, "Verified Boot," http://www.chromium.org/chromium-os/chromiumos-design-docs/verified-boot, online, Accessed December 2012.
[22] A. Nagarajan, V. Varadharajan, M. Hitchens, and E. Gallery, "Property based attestation and trusted computing: Analysis and challenges," in *Proceedings of the 2009 Third International Conference on Network and System Security*, ser. NSS '09. Washington, DC, USA: IEEE Computer Society, 2009, pp. 278–285.
[23] J. M. McCun, "Reducing the trusted computing base for applications on commodity systems," Ph.D. dissertation, School of Electrical and Computer Engineering, Carnegie Mellon University, 2009.
[24] C. Kil, E. C. Sezer, A. M. Azab, P. Ning, and X. Zhang, "Remote Attestation to Dynamic System Properties: Towards Providing Complete System Integrity Evidence," in *Proceedings of the 39th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN 2009)*, 2009, pp. 115–124.
[25] L. Davi, A.-R. Sadeghi, and M. Winandy, "Dynamic integrity measurement and attestation: towards defense against return-oriented programming attacks," in *Proceedings of the 2009 ACM workshop on Scalable trusted computing*, ser. STC '09. ACM, 2009, pp. 49–54.
[26] A. Leicher, N. Kuntze, and A. U. Schmidt, "Implementation of a trusted ticket system," in *SEC*, ser. IFIP, D. Gritzalis and J. Lopez, Eds., vol. 297. Springer, 2009, pp. 152–163.
[27] A. Pashalidis and C. Mitchell, "Single sign-on using trusted platforms," in *Information Security*, ser. Lecture Notes in Computer Science, C. Boyd and W. Mao, Eds. Springer Berlin Heidelberg, 2003, vol. 2851, pp. 54–68.
[28] A. Leicher, A. Schmidt, Y. Shah, and I. Cha, "Trusted Computing enhanced OpenID," in *Internet Technology and Secured Transactions (ICITST), 2010 International Conference for*, nov. 2010, pp. 1 –8.
[29] W. Hu, H. Tan, P. Corke, W. C. Shih, and S. Jha, "Toward trusted wireless sensor networks," *TOSN*, vol. 7, no. 1, pp. 5:1–5:25, 2010.
[30] C. Krauß, F. Stumpf, and C. M. Eckert, "Detecting node compromise in hybrid wireless sensor networks using attestation techniques," in *ESAS*, 2007, pp. 203–217.
[31] Y. M. Yussoff, H. Hashim, and M. D. Baba, "Identity-based trusted authentication in wireless sensor network," *CoRR*, vol. abs/1207.6185, 2012.
[32] A. Seshadri, M. Luk, A. Perrig, L. van Doorn, and P. K. Khosla, "Scuba: Secure code update by attestation in sensor networks," in *Workshop on Wireless Security*, 2006, pp. 85–94.