Trustworthy Communication Channels for the Electronic Safe

Christian Breitenstrom[2], Martin Unger[1] and Andreas Penski [2]
[1]Technical University, Berlin, Germany
[2]Fraunhofer Institute FOKUS, Berlin, Germany
martin.unger@cs.tu-berlin.de
christian.breitenstrom@fokus.fraunhofer.de
andreas.penski@fokus.fraunhofer.de

Abstract: The transition from the traditional task oriented delivery of single government services to a process-oriented delivery of well sized service bundles is common for modern eGovernment implementations. Concerning the European Services Directive (DIRECTIVE 2006/123/EC) these service bundles are tailored to cover the needs of companies trying to offer their services in other EU member states. Nevertheless this approach is applicable to many other life circumstances. As process oriented service chains require well suited data capture and data sharing mechanisms, the Electronic Safe as a fundamental eGovernment infrastructure comes into play. The Electronic Safe is seen from a citizen's viewpoint as a unique instrument, that makes it easy to control data flows between the citizen and her communication partners in a trustworthy and privacy conserving way. Previous work showed how to implement such a critical infrastructure in a decentralized and distributed manner, to satisfy strong confidentiality and privacy requirements. This paper focuses on the ability to bind the functionality of the Electronic Safe to the trustworthiness of its underlying hardware and software stack including its secure communication channels. We show how to use the mutual attestation mechanisms designed by the Trusted Computing Group (TCG) with the Safe Infrastructure and their communication protocols, while keeping the privacy features that the Safe Owner appreciates.

Keywords: trusted computing, electronic safe, privacy enhancing technology

1. Introduction

1.1 Privacy patterns

One of the key patterns used to avoid the tracking of the user activities in the internet is to implement proxy mechanisms wherever possible. Outside the digital world hard cash is the privacy preserving way of spending money. It can be considered as a proxy mechanism for bank transactions. Instead of using the banking service and disclose the information, which amount of money is spent for what service we use coins until we have to contact the bank again. Increasingly this gets feasible with digital assertions of claims that we present to use electronic services. Often we have no choice as to disclose personal information about us for authorization. The needed assertions can be stored in an Electronic Safe [42, 43, 44] for later use. So the Electronic Safe is used as a proxy to prevent traceability.

1.2 Electronic Safe as trustworthy infrastructure

The Electronic Safe consists of a collection of components and services that let the Safe Owner store her digital content in a privacy enhanced, secure way at a number of independent ISPs called Storage Providers. The way of the data splitting implicates redundancy to cope with availability problems and assures that no one of the Storage Providers receives enough data to recover any clear text. Even if a limited number of Storage Providers collude, they have no chance to determine the data pieces belonging to one document. As we are working with anonymous credentials, the legitimate Owner of a data piece can prove its ownership without revealing its identity.

1.3 Untrustworthy communication partners

Given such an infrastructure, the Safe Client is the weakest link in the chain between the Safe User and the Safe Services. An adversary could infect the user's desktop to read all the stored information in clear text, when the user is editing her data. Even within managed environments the user can't be sure that the antivirus program detects all malware on her desktop. If the antivirus program doesn't detect any problems it could be interpreted in more than one way: (1) the PC is clean, (2) it is infected

Reference this paper as: Breitenstrom, C, Unger, M, and Penski, A. "Trustworthy Communication Channels for the Electronic Safe" *Electronic Journal of e-Government* Volume 8 Issue 2 2010, (pp93-104), available online at www.ejeg.com

by unknown malware or (3) the antivirus itself is modified. It is even less probable that the user will be able to judge the trustworthiness of the Safe Service.

1.4 Critical acceptance factors

The internet user has currently no objective means to measure the trustworthiness of used services. The trust in existing services results from a range of independent factors like perceived risk, perceived usefulness, ease of use, the reputation of the service provider and the satisfaction with past transactions. In a recent survey among 65 people visiting a nightly Science event at the Technical University Berlin that were in particular interested in the new German eID-Card we asked about their trust in different data protection seals from independent non-governmental institutions. It appeared that more than half of the participants do not trust these data protection and audit seals, which is one of the key arguments to sell cloud computing services. It can be assumed that the recurring news about data leakages even in public sector institutions and about successfully hacked ID cards, contribute to this result.

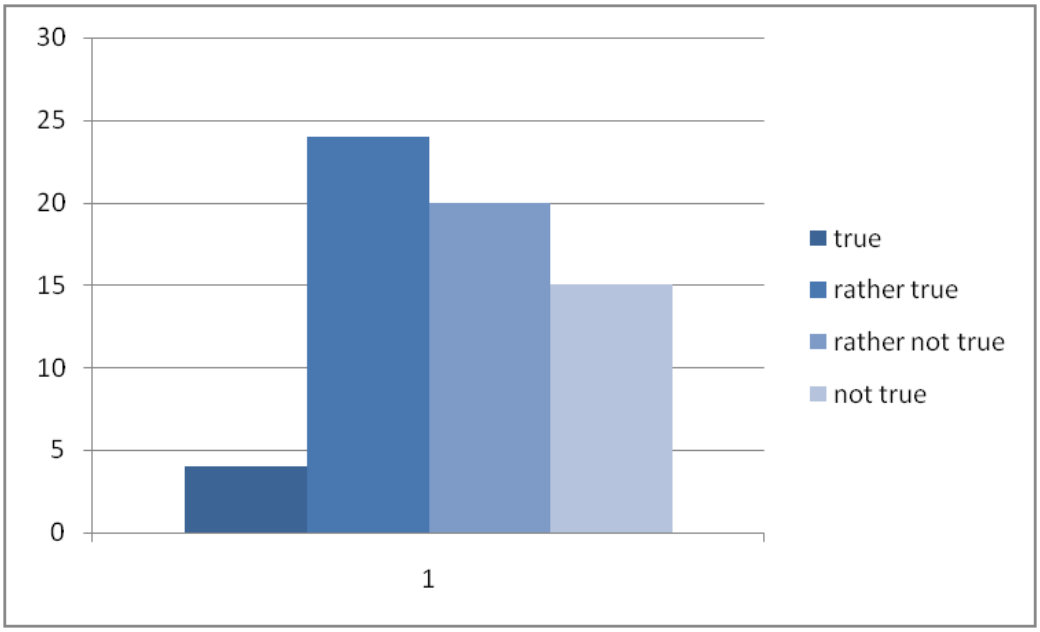

Figure 1: Thanks to the privacy seal (TÜV, Privacy Foundation, WOT) I can be sure to avoid fraud

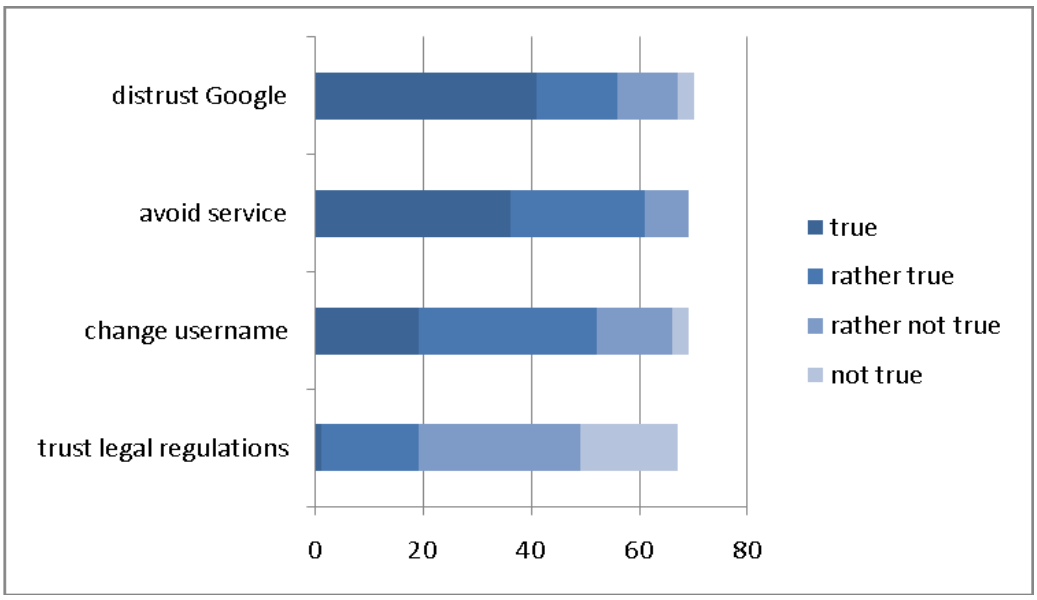

Figure 2: User's reaction on current privacy concerns

Some other questions reveal the need for some profound real time metrics for the trustworthiness of services. More than half of the people are concerned about the knowledge that Google gets from their

search activities, about half of the people avoid services that require personal information as authorization means. Less than 30% of these people feel protected by the current data protection laws. About the same number of people regularly use different usernames (pseudonyms) to avoid the linkage of their activities at different service providers. As these numbers are captured among the most active, technology affine people which act as early adopters of Electronic Safes, we deduce that substantial improvements in the way how the Safe User is supported to judge whether his own desktop PC and the Safe Service is trustworthy or not, will significantly increase the acceptance of the Electronic Safe at all.

1.5 Binding the communication channel to a trustworthy state

Most of the security sensitive applications on the Internet (e.g., online banking, eCommerce and e-government) typically deploy secure channels such as TLS [3] or IPSec [13] to provide secure access to and communication with the corresponding services. These security protocols protect data during transmission and allow to authenticate the endpoints.

However, they do not provide any protection from (maliciously) modified software running on an endpoint. More precisely, setting-up a secure channel is currently not linked to the integrity of an endpoint. Nevertheless, most attacks concern compromising the endpoints by injecting malicious code rather than compromising the secure channel.

Thus, for the secure provision of Safe services over the Internet endpoint integrity is vital. To avert such attack scenarios, information on the communication endpoints integrity or configuration has to be provided in a secure and reliable manner, to enable the peers to judge each other's trustworthiness based on the information received.

Reporting integrity information of a remote platform in a reliable way is one of the main goals of Trusted Computing (TC) as proposed by the Trusted Computing Group (TCG, [31]). The basic idea is to securely capture configuration information of the core components of the platform (firmware and software). This information is stored in a cost-effective, tamper resistant Trusted Platform Module (TPM). The TPM in turn is mounted on the mainboard of the computing platform and acts as trust anchor. It can sign gathered configuration information and report it to a requesting party. This process is called attestation by the TCG. Additionally data can be stored bound to a specific platform configuration, i.e. the data are only available in plain text, if the systems works exactly in that configuration. The TCG calls this mechanism binding/sealing data.

In our approach we use a combination of TCG TC functionalities and the TLS protocol to form a Trusted Channel. A proof-of-concept implementation of this concept is described in [39].

The central feature of the Trusted Channel is the capability to provide reliable evidence concerning the trustworthiness of a communication partner. Furthermore, by means of a specific system architecture we are able to enforce the security of data not only during transmission but also on the involved endpoints. It has to be pointed out that the linkage of configuration information to the TLS channel is crucial to prevent relay attacks where the configuration of a third platform, deemed trustworthy, is relayed by an attacker, acting as Man-in-the-Middle (MitM).

We use the Trusted Channel approach to provide a secure communication channel between the Safe Provider and the Safe User.

2. Security requirements

These are the properties that have to be fulfilled to meet the security requirements necessary in a Safe scenario

(SR1) **Secure channel properties**: Integrity and confidentiality of data, freshness to prevent replay attacks, and authenticity both during transmission as well as within the endpoints have to be provided.

(SR2) **Authentic linkage of configuration/integrity information to the secure channel**: Authentic configuration/integrity information must be bound to the trusted channel (i.e., during the establishment and while the Trusted Channel is in place) to prevent relay attacks.

(SR3) **Privacy**: Creation and maintenance of the channel should adhere to the least information paradigm, i.e., disclosure of a platform's configuration/integrity information not beyond what is necessary for proper integrity validation. Furthermore, platform configuration information has to be protected against disclosure to a third party.

3. Functionalities and mechanisms used in our approach

The configuration of a platform is represented by a combination of credentials vouching for security relevant properties of the platform's components (hardware and/or software). Deriving those properties can be done in different ways [20, 9, 6]. The TCG proposes to compute SHA1 [4] hash values over code (software/firmware) for that purpose. The mechanism of deriving these hash values is called measurement. These hash values are designated as digital fingerprints, since they are used to unambiguously identify components.

To be able to derive the trustworthiness of a platform we have to compare the digital fingerprints reported by a counterpart to reference values. In our approach reference values represent digital fingerprints provided and signed by a Trusted Third Party (e.g., the distributor of the Safe Client). X.509 Certificates are used for authentication purposes.

The communication endpoints of our implementation operate based on compartments. A compartment consists of one or a group of software components that is logically isolated from other software components. Isolation means that a compartment can only access data of another compartment using specified interfaces provided by a controlling instance.

The set of all security critical software and hardware components of a platform responsible for preserving its trustworthiness is called Trusted Computing Base (TCB). Thus, it is crucial to keep the TCB isolated and as small as possible to avoid known problems and vulnerabilities arising along with code complexity. The TCB ideally is protected using execution isolation technologies like Intel TXT to avoid any manipulation.

The central component of the TCB is formed by the TPM, which is currently implemented as a dedicated hardware chip. It offers amongst others a cryptographic hash function (SHA-1), a cryptographic engine (RSA) for encryption/decryption and signing, a hardware-based Random Number Generator (RNG), hardware protected monotonic counters as well as some amount of protected storage. It provides a set of registers in protected storage called Platform Configuration Registers (PCR) that can be used to store hash values. Protected storage is also used to store certain security sensitive keys, e.g., Attestation Identity Keys (AIKs) or the Storage Root Key (SRK).

3.1 Important TPM-keys and their usage

An AIK is non-migratable, i.e., its private part never leaves the TPM's protected storage and can only be applied for signing data (e.g., via the TPM Quote()-command) that originates from the TPM but not for encryption. An AIK certificate ($cert_{AIK}$) that vouches for the mentioned AIK properties can be obtained from a Certification Authority (CA). AIKs can be used to reliably authenticate users and/or systems. They are kept securely inside the TPM and can only be used to authenticate stored measurement values in an attestation or certifying other non-migratable keys [31]. The SRK is kept inside the TPM as root for the whole key hierarchy [31].

The last important Key that has to be mentioned is the Endorsement Key (K_{END}). K_{END} is a unique key introduced into the TPM during the manufacturing process and never leaves the TPM. It guarantees that the TPM adheres to the TCG specifications and thus vouches for the security features of the TPM. For this purpose an Endorsement Certificate ($cert_{END}$) is applied. Further, the K_{END} is used to sign fresh AIKs, stating that they are created in a secure environment, as specified by the TCG. This enables a CA to certify the security of this new AIK because the signature by K_{END} allows the CA to assess the trustworthiness of its creation and storage.

To improve security of the common TLS protocol, we move all security relevant operations like, e.g., encryption, signing and the handling of credentials to the TCB, whose code is protected against a wide range of attacks running in separate memory space and only accessible via interfaces. The protocol implementation remains in user space, because there is no need to protect it.

4. Basic system architecture

Our system architecture is based on security frameworks as proposed, e.g., in [21], [23], and consists of Application, Trusted Service, Virtualization as well as a Hardware Layer. We kept our approach generic, thus it is possible to implement/integrate the components in different systems also on common operating systems like, e.g., Linux or Windows. But, if these monolithic OSs are applied, some constraints have to be considered when looking at the security of such implementations, because in general they are not capable to ensure strong isolation of processes and corresponding data.

Hardware Layer: The hardware layer has to offer TC extensions that are conform to the relevant TCG specifications (e.g., [31]). This essentially means that it comprises a TPM chip and a compatible BIOS. For better performance we recommend using modern virtualization features like Intel VT [41]. For stronger isolation of the Safe application mechanisms like Intel TXT [40] should be used.

Virtualization Layer: The virtualization layer offers and mediates access to central hardware components like, e.g., CPU and MMU. These tasks can be performed by many kinds of virtualization techniques, namely hypervisors, microkernel approaches or a common OS running a virtualization application, e.g., VMware [37].

Trusted Service Layer: This layer consists of security services and provides interfaces to the Application Layer. It also mediates and monitors access to virtualized hardware resources. Subsequently, we briefly describe the main components of the TCB in our approach:

- **Trust Manager** (TM) provides functionalities used for establishing Trusted Channels. To be able to provide this functionality TM bundles multiple calls to the TPM into a simple API for calling instances. Thus, it offers functionality to generate keys, bind/unbind, seal/unseal, certify these keys or to report the current measurement values of the TCB stored inside the TPM. The keys used in the initial handshake are computed and held by the TM . They never leave the TCB.
- **Compartment Manager** (CM) is responsible for starting and stopping compartments. It measures the compartment code when starting it and assigns a locally unique ID to this compartment. This ID as well as the measurements are reported to the Integrity Manager.
- I**ntegrity Manager** (IM) stores the compartment's properties. In our approach this means appending the measurements reported by CM together with a unique ID to a Configuration Data Structure (CDS). The CDS itself holds a list of measurements of the binary images of all client compartments running on top of the TCB. IM keeps the CDS secure by storing it inside the TCB's memory space and provides it to other TCB components.
- **Storage Manager** (SM) handles persistent data storing for the different compartments.

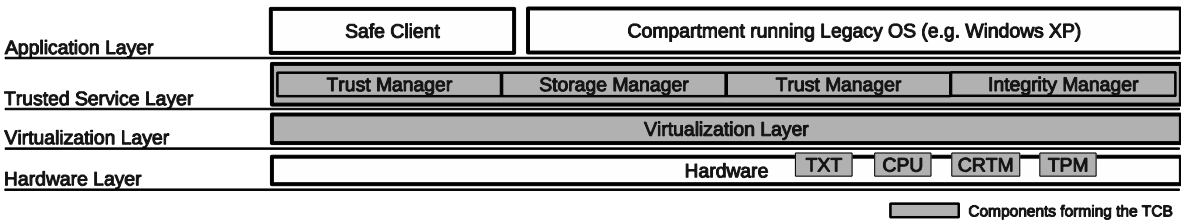

Figure 3: Basic architecture for the trustworthy safe system

Application Layer: In this layer the applications run in isolated compartments. This can be either applications running directly on top of the underlying TCB or whole OSs. In our scenario we run a Safe Client or Server compartment in parallel to a legacy OS like Windows.

5. Monitored start-up process

To be able to attest a platform's configuration, its hard- and software components are measured reliably and those measurements are stored securely. That means a hash value is computed over every piece of software before it is started. An ongoing measurement process is effected originating from the Core Root of Trust for Measurement (CRTM) that initiates the measurement process up to and excluding the Application Layer. The CRTM is a small piece of code initiating the measurement process at the very beginning of the boot process. Usually, this code is located within the BIOS.

Using security features like Intel TXT this measurement process can also be started while the system is already running. At the moment a sensitive application like our Safe application is executed, the currently running OS is halted and all interrupts are blocked. Then a software component called **dynamic** CRTM is started and loads the software parts of the TCB from protected memory. After that a compartment is loaded containing the Safe application running on top of the TCB that has been just loaded.

Consequently, a Chain of Trust (CoT) is established and the TCB is measured reliably. These measurements are stored inside the TPM and represent the **static configuration** in our approach, because it must be only modifiable with a subsequent reboot. After the boot process platform monitoring is conducted by CM . Thus, CM extends the CoT whenever a client compartment is loaded that runs on top of the TCB. The CDS that reflects the platform configuration is maintained by the IM. The configuration of compartments that run above the TCB represents the **dynamic configuration**.

6. Description of attestation data structures

In this section we introduce credentials and extensions to TLS handshake messages we use to set up a Trusted Channel.

6.1 TLS Key Exchange and Certificate Elements

In a common TLS handshake certificates are used to authenticate the peers. The certificate extensions and credentials we define are necessary to bind the TLS channel to the endpoints whose configuration is reported, and to be capable of proving that a certain TCB is in place.

These new credentials are held in an environment protected by TC mechanisms and are loaded at start-up. Therefore, we act on the assumption that these additional security measures justify stronger security assumptions concerning the storage and usage of those credentials. In the following paragraphs we explain their creation, storage and interdependency.

- **SKAE Key (K_{SKAE})[1] and SKAE**: The non-migratable[2] asymmetric key pair K_{SKAE} (PK_{SKAE}, SK_{SKAE}) is created after an AIK (K_{AIK}) has been certified and installed. Its private part SK_{SKAE} is sealed to a specific TCB using the SRK ($K_{storage}$) and never leaves the TPM unencrypted. We make use of the Subject Key Attestation Evidence (SKAE) as proposed by the TCG [29]. In contrast to the intended purpose, we are able to use the SKAE as standalone element within our handshake, but we also foresee the possibility to include it in a X.509 certificate as it is used for authentication purposes in the communication with the Safe provider. The SKAE basically consists of a TPM_CERTIFY_INFO2 structure representing the TCB configuration that has to be in place during key release (including a digest of PK_{SKAE} [33, p.96]) and a signature over this structure (Sig_{SKAE}) by a K_{AIK}. Additionally, links to reference values can be provided. The SKAE can vouch that K_{SKAE} was created by a Trusted Platform that conforms to the TCG specification [31] and that a certain TCB configuration has to be in place during release, because of the sealing mechanism.
- **Secure Signature Key (K_{sign})**: We introduce the asymmetric key pair Secure Signature Key K_{sign} (PK_{sign}, SK_{sign}), that are considered long-lived and usable for client compartments that wish to establish a Trusted Channel to a remote party, here the Safe Provider. They are created inside the TCB and sealed using $K_{storage}$.

K_{sign} is included in the common TLS certificate ($cert_{TC\text{-}TLS}$) as signature key. Thus, the public part of K_{sign}, $Pk_{sign,}$ is put into the public key field of the X.509 certificates used to authenticate citizen, requester or Safe provider, respectively. The signature key is needed during the handshake to provide the **binding between integrity information and the endpoints**.

Therefore, the usual TLS authentication scheme will be used for the authentication of the Safe Provider server and its $cert^{S}_{TC\text{-}TLS}$ will likely be signed by a CA like e.g. Verisign. The TLS client authentication mechanism, optional for a standard TLS channel, must be used to guarantee the binding through $cert^{C}_{TC\text{-}TLS}$ as well as the authentication of the client. Therefore, we use a certificate that is either signed by a CA that is believed to be trustworthy in the eyes of the Safe provider or it is

1 We denote Keys with the letter K, its secret parts with SK and its public parts with PK, respectively. Signature Keys are given the subscript sign and enc is the subscript marking encryption keys. The superscripts S and C stand for server or client. That means SK^{S}_{sign} designates the secret part of a signature key of the server. The shortcut cert stands for certificates.
2 The private part of an asymmetric key pair labeled non-migratable never leaves the TPM unencrypted [31].

signed by the Safe provider himself. If authentication should not be necessary the certificate can also be self-signed.

K_{sign} is unsealed by TM during start-up. K_{SKAE} is used to authenticate K_{sign} by signing its public part. It would also be possible to use K_{SKAE} instead of K_{sign} for signing during the handshake. But then the involvement of the TPM every time a Trusted Channel is set up would be necessary. This would result in a significant performance loss, especially in connection with server systems. This is why we introduced the intermediary K_{sign}. Using K_{AIK} directly for signing bulk data is not allowed by the TCG specification. By signing PK_{sign} using SK_{SKAE} we provide twofold evidence: that the TCB identified by the SKAE was in place during the signature and it is a statement from that TCB about K_{sign} like: "I certify that Ksign is correctly treated, i.e., when decrypted, the keys are kept secret by myself". If the verifier of the SKAE trusts the TCB attested by it, then the verifier can also trust the TCB's statement about the correct treatment of K_{sign} . Therefore, the TPM_Sign() function is applied by TM to sign K_{sign} 's public part with SK_{SKAE} at boot time. The resulting signature Sig_{PKsign} is also held by TM. TM now holds K_{sign}, PK_{SKAE} and Sig_{PKsign}. These credentials are kept inside the TCB.

6.2 Extensions used in the TLS handshake

The extensions to the TLS protocol that will be introduced in the following paragraphs are necessary to trigger and negotiate the exchange of configuration information as well as for the transport of the additional configuration/integrity data. Extensions to the TLS handshake protocol can be small data chunks added to the Hello messages [3] or completely new handshake messages. These extensions are explicitly foreseen by the TLS specification to deal with advancements and changes in communication infrastructures. There already exist several extensions to the TLS protocol, e.g., for sending client certificate URLs or explicit server name indication [1]. The basic concept for extending TLS with an additional handshake message is described in RFC4680 available from the Internet Engineering Task Force (IETF) Networking Group [25]. This RFC defines the additional SupplementalData handshake message envisioned to carry additional generic data, whose format must be specified by the application that uses it, and whose delivery must be negotiated via Hello message extensions.

Hello message extensions: Attestation Extension (AttestExt) is transmitted within the ClientHello and ServerHello messages, respectively. The first one is used in the initial handshake to negotiate which side (C and/or S) has to attest to its state and the type of attestation supported or if privacy of configuration information is desired.

Supplemental Data Message Creation and Evaluation: The SupplementalData message includes the SKAE, PK_{SKAE} , $cert_{AIK}$, Sig_{PKsign}, a concatenation of the nonces sent in the TLS Hello messages and measurements representing the image of the client compartment that runs the Safe Client. These measurements are extracted from the CDS and corresponding signed reference values or links to them are provided. Furthermore, a Signature $Sig_{AttestData}$ on AttestData is appended. This signature is needed to bind the AttestData structure to the respective secure channel endpoint. In Figure 4 we show how AttestData is composed.

Determination of endpoint trustworthiness: The trustworthiness of the peer's TCB is determined by evaluating Sig_{SKAE} and the TCB measurement included in SKAE using reference values provided by trusted third parties. To verify the validity of K_{sign}, Sig_{PKsign} is checked. Then the linkage between secure channel endpoint and AttestData is verified by inspecting $Sig_{AttestData}$. Freshness of AttestData is guaranteed comparing nonceSD to the nonces sent the hello messages. Finally, the trustworthiness of compartments running on top of the TCB is determined in a next step by evaluating measurements using either additional reference values provided by the peer within the measurements data field or by trusted third parties.

Since TLS handshake messages are usually sent in cleartext, in case privacy of attestation information is desired by one of the communication partners, no SupplementalData messages are sent within the first handshake. Subsequently, a second handshake is performed directly after the first one to exchange attestation information encrypted using the session key negotiated in the previous handshake [25].

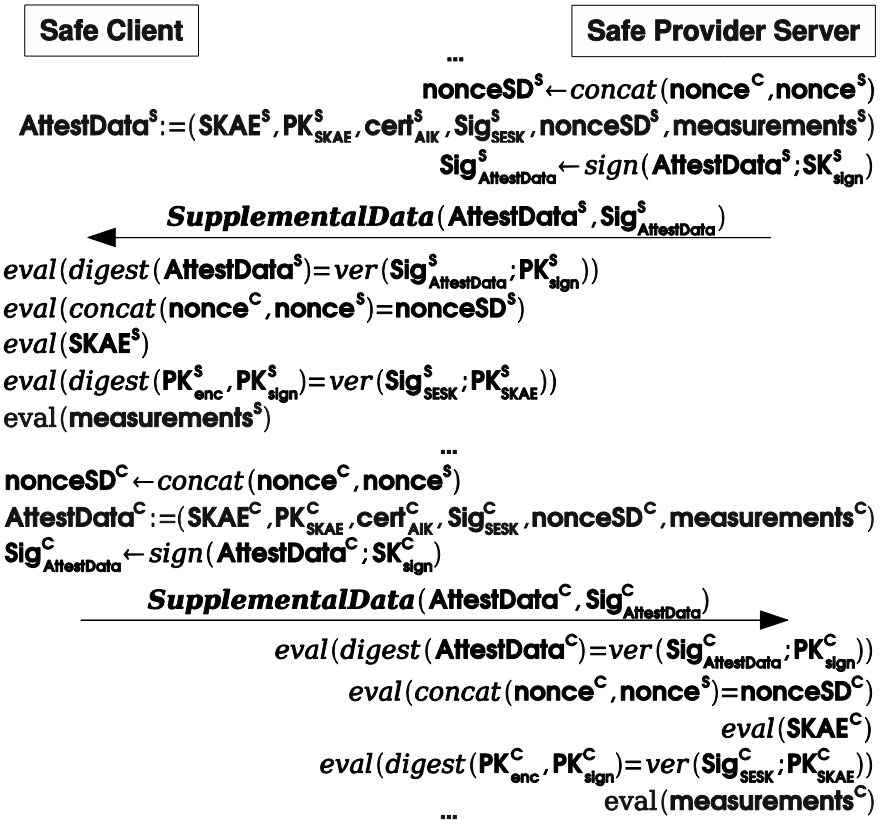

Figure 4: Creation and evaluation of the SupplementalData message

7. Setting up a trusted channel to a safe provider using TLS

After the system has booted up and the TCB is in place, the TM is used to unseal secrets like signature or encryption keys and certificates. Thus, the system is initialized and ready to set up a Trusted Channel.

This is done in following these steps:

- **Negotiating Security Parameters**: To set up a Trusted Channel, the Client (C, in our case the citizen or a requester desiring to gain access to an Safe) starts the Safe Client application as well as the TLS software and sends a ClientHello message to the Safe provider server (S) that answers with a corresponding ServerHello message. Using those hello messages the two parties involved in the communication negotiate the attributes of the Trusted Channel they want to establish. In contrast to the common TLS design the nonces sent in the hello messages are taken from a RNG seeded by the TPM.
- **Configuration and Key Exchange**: Each peer provides evidence related to its configuration and integrity. For this purpose we use additional SupplementalData messages as defined in Internet Engineering Task Force (IETF) RFC4680 (cf. Section 5, [25]). Thus, SupplementalData messages are composed to transfer AttestData representing configuration/ integrity information. AttestData is signed using a secret key (SK_{sign}) for authentication and integrity protection of configuration information. Following the SupplementalData message, each side provides a certificate ($cert_{TC\text{-}TLS}$) including the respective public key (PK_{sign}) used to verify this signature.

Subsequently, keys are computed on both sides. The public Diffie-Hellman values used for TLS Session Key (SeK) computation are signed using SK^S_{sign} to provide authentication evidence. S then sends DH^S_{public} and a signature (Sig^S_{DH}) to C within the ServerKeyExchange message. C computes its own values and sends DH^C_{public} to S using the ClientKeyExchange message. The following CertificateVerify message is used to prove the possession of SK^C_{sign}, and to authenticate DH^C_{public}, by signing a digest over all previously exchanged handshake messages (prev) using SK^C_{sign} [3].

- **Session Key Computation**: Following CertificateVerify, the TLS master secret (ms) is computed on both sides using $nonce^C$, $nonce^S$, a string indicating that this is a ms, and the result of the final Diffie-Hellman computation as input to a pseudo random function (PRF). Subsequently, the

SeK is derived from the ms on both peers [3, p.24]. At last, the handshake is finalized by the ChangeCipherSpec protocol and final Finished messages. These Finished messages are already encrypted using SeK, thus, a failure in key exchange would be noticed.

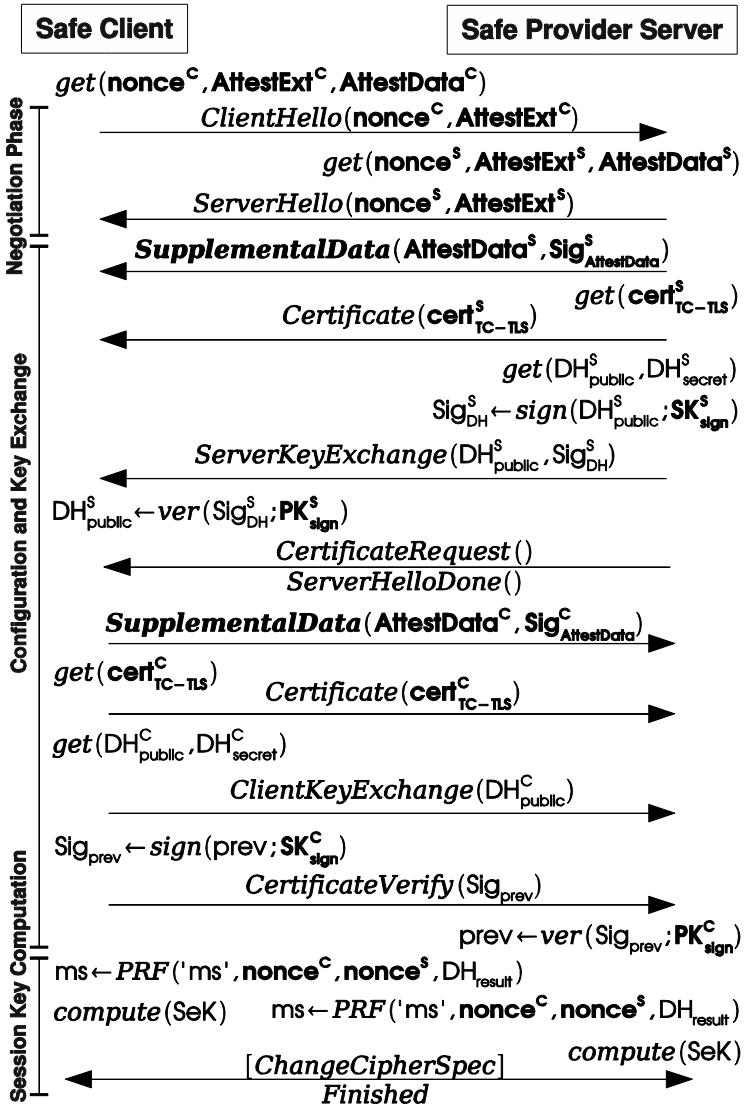

Figure 5: The phases of the adapted TLS handshake

8. Security considerations

In this section we carry out a short evaluation of the security requirements presented:
(SR1) **Secure channel properties**: TLS provides secure channel properties during transmission. On the endpoints the TCB offers those properties. Confidentiality and integrity are provided by trusted initialization, isolation of the TCB and platform monitoring. The concept is even able to provide protection against a malicious administrator, because the measurements of the TCB cannot be faked, and if the TCB is properly configured (expressed by its measurements), it should not be possible to tamper with the TCB while running. The TCB also takes care of authenticity and freshness by securely storing nonces and session keys. As a result of platform monitoring, every manipulation of a compartment is noticed and access to sensitive data can be barred if necessary to ensure the security properties. Furthermore, SM provides storage that can preserve secure channel properties in case that data is stored persistently.

(SR2) **Authentic linkage of configuration information and secure channel**: Authenticity of communication is guaranteed by providing $cert_{TC\text{-}TLS}$ that is used to authenticate the endpoints. The secure linkage of configuration information to the endpoints is verified by evaluating the SKAE, Sig_{PKsign} and $Sig_{AttestData}$.

We assume a secure as well as specification conform creation of $K_{storage}$ (Storage Root Key) and AIK. We further assume that a TCB whose configuration has been evaluated by the counterpart is able to reliably transfer configuration information related to the client compartments and takes care of the secure storage and application of the keys used within the handshake. A possibility for the retrieval of CA keys for verification of signatures is also anticipated.

After a successful evaluation of the credentials transferred, these statements can be made: All keys are bound to the same TCB. This TCB is specified by measurement values incorporated in SKAE. Thus, the measurements of the Safe application image sent, originate from this TCB because SK_{sign} is sealed to this TCB and signed by SK_{SKAE}.

SeK and K_{sign} are kept inside the TCB during the whole session. Due to the isolation property of the TCB those keys cannot be disclosed to compartments or to other platforms. Relay attacks as well as attacks to obtain any keys establishing the Trusted Channel are not feasible assumed that no hardware attacks are applied. For the full certificate chain see Figure 6.

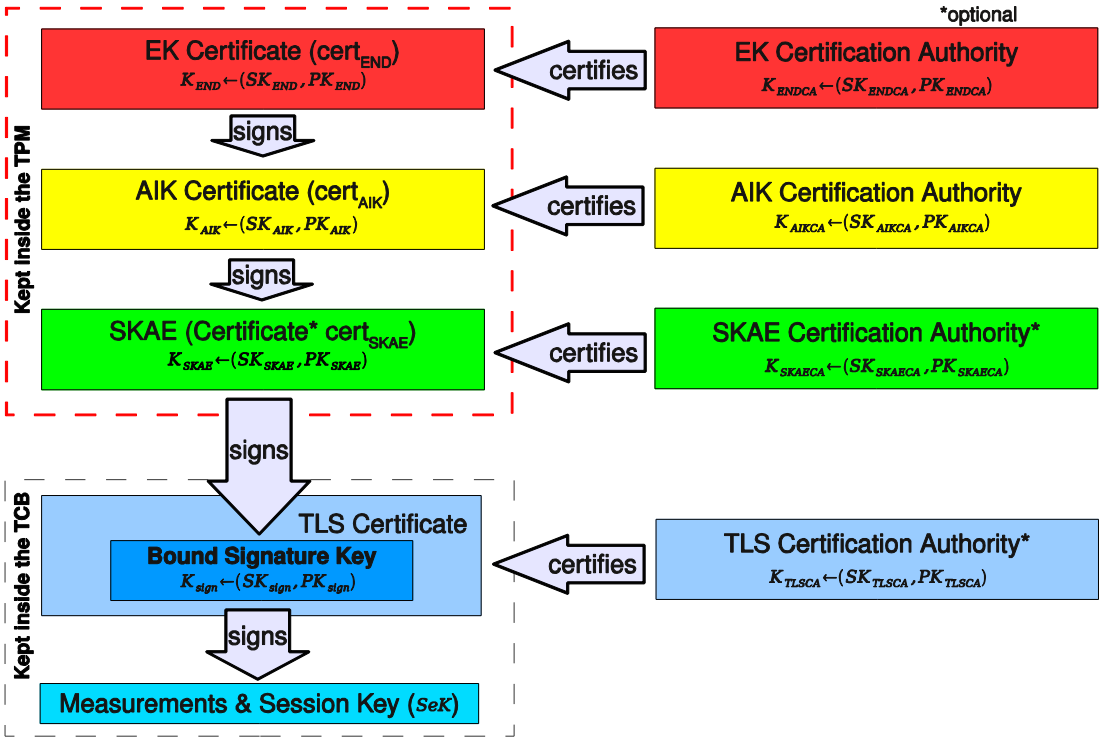

Figure 6: Certificate chain that vouches for the secure linkage of endpoints to the trusted channel

(SR3) **Privacy**: With regard to configuration data transmitted, we provide a possibility to send it encrypted to protect potentially sensitive data. Only the configuration of TCB and the Safe Client compartment is reported to the peer, keeping the information disclosed to the other platform as minimal as possible. Furthermore, every communication partner can decide on the trustworthiness of its counterpart and thus, decide whether it will proceed to the next step of the communication protocol.

References

Armknecht, F., Gasmi, Y., Sadeghi, A., Stewin, P., Unger, M., Ramunno, G., Vernizzi, D. An Efficient Implementation of Trusted Channels based onOpenSSL, ACM-STC'08, 2008.

Blake-Wilson, S. et al. Transport Layer Security (TLS) Extensions. IETF RFC 4366, Apr. 2006.

Breitenstrom, C., Brunzel, M. Klessmann, J.: White Paper Elektronische Safes für Daten und Dokumente, Fraunhofer FOKUS, , http://www.fokus.fraunhofer.de/de/elan/_docs/_hpp-gruppe/esafe_white-paper_081219.pdf, 2008

Breitenstrom, C., Penski, A.: Electronic Safes for Process Oriented eGovernment, ECEG, 2010

Chess, D., Dyer, J., Itoi, N., Kravitz, J., Palmer, E., Perez, R., and Smith, S. Using trusted co-servers to enhance security of web interaction, Mar. 2007.
Dierks, T., and Rescorla, E. The Transport Layer Security (TLS) Protocol Version 1.1. IETF RFC 4346, Apr. 2006.
Eastlake, D., and Jones, P. US Secure Hash Algorithm 1 (SHA1). IETF RFC 3174, Sept. 2001.
EMSCB Project. Towards Trustworthy Systems with Open Standards and Trusted Computing, 2008.
Franz, M., Chandra, D., Gal, A., Haldar, V., Probst, C. W., Reig, F., and Wang, N. A portable virtual machine target for proof-carrying code. Journal of Science of Computer Programming 57, 3 (sep 2005), 275-294.
Gasmi, Y., Sadeghi, A.-R., Stewin, P., Unger, M., and Asokan, N. Beyond Secure Channels. ACM STC '07 Proceedings, 2007.
Goldman, K., Perez, R., and Sailer, R. Linking remote attestation to secure tunnel endpoints. STC '06 Proceedings, Nov. 2006.
Haldar, V., Chandra, D., and Franz, M. Semantic remote attestation: a virtual machine directed approach to trusted computing. VM '04 Proceedings, 2004.
Housley, R., Ford, W., Polk, W., and Solo, D. Internet X.509 Public Key Infrastructure Certificate and CRL Profile 5. IETF RFC 2459, Jan. 1999.
IETF. SSL 3.0 specification, May 2008.
Intel Corp., Intel Trusted Execution Technology Architectural Overview, 2003
Intel Corp., Intel Virtualization Technology for Directed I/O - Architecture Specification, September 2008
Jiang, S., Smith, S., and Minami, K. Securing Web Servers against Insider Attack. ACSAC '01 Proceedings, 2001.
Kent, S., and Seo, K. Security Architecture for the Internet Protocol. IETF RFC 4301, Dec. 2005.
Kuhlmann, D., Landfermann, R., Ramasamy, H., Schunter, M., Ramunno, G., and Vernizzi, D. An Open Trusted Computing Architecture – Secure virtual machines enabling user-defined policy enforcement, 2006.
Leech, M., Ganis, M., Lee, Y., Kuris, R., Koblas, D., and Jones, L. SOCKS Protocol Version 5. IETF RFC 1928, Mar. 1996.
Marchesini, J., Smith, S. W., Wild, O., Stabiner, J., and Barsamian, A. Open-Source Applications of TCPA Hardware. ACSAC '04 Proceedings, 2004.
OpenSSL Project. The OpenSSL Project Homepage, 2007.
OpenTC Project. The OpenTC Project Homepage, 2008.
Penski, A., Breitenstrom, C.: Der elektronische Safe für Daten und Dokumente, DACH Security, 2010
RSA laboratories. Public-Key Cryptography Standards (PKCS), May 2008.
Sadeghi, A.-R., and St"uble, C. Property-based attestation for computing platforms: caring about properties, not mechanisms. NSPW '04 Proceedings, 2004.
Sadeghi, A.-R., Stüble, C., and Pohlmann, N. European multilateral secure computing base – open trusted computing for you and me. Datenschutz und Datensicherheit DuD 28, 9 (2004), 548-554. Verlag Friedrich Vierweg & Sohn, Wiesbaden.
Sadeghi, A.-R., Stüble, C., Wolf, M., Asokan, N., and Ekberg, J.-E. Enabling Fairer Digital Rights Management with Trusted Computing. ISC '07 Proceedings, 2007.
Sailer, R., Valdez, E., Jaeger, T., Perez, R., van Doorn, L., Griffin, J. L., and Berger, S. sHype: Secure hypervisor approach to trusted virtualized systems. Techn. Rep. RC23511, Feb. 2005. IBM Research Division.
Sailer, R., Zhang, X., Jaeger, T., and van Doorn, L. Design and implementation of a TCG-based integrity measurement architecture. SSYM'04 Proceedings, 2004.
Santesson, S. TLS Handshake Message for Supplemental Data. IETF RFC 4680, Sept. 2006.
Selhorst, M. TrustedGRUB - Details, Apr. 2008.
Squid-cache.org. Squid: Optimising Web Delivery, May 2008.
Stumpf, F., Tafreschi, O., Röder, P., and Eckert, C. A robust Integrity Reporting Protocol for Remote Attestation. WATC '06, Dec. 2006.
TCG Infrastructure Working Group (IWG). TCG Infrastructure Workgroup Subject Key Attestation Evidence Extension, June 2005. Specification Version 1.0 Revision 7.
TCG Infrastructure Working Group (IWG). TCG Infrastructure Working Group Reference Architecture for Interoperability (Part I), June 2005. Specification Version 1.0 Revision 1.
Trusted Computing Group. TCG Specification Architecture Overview, Mar. 2003. Specification Revision 1.3 28th March 2007.
Trusted Computing Group. TCG Software Stack (TSS) Specification Version 1.2, Jan. 2006. Specification Version 1.2 Level 1 Final.
Trusted Computing Group. TCG TPM Main Part 2 TPM Structures, Mar. 2006. Specification Version Level 2 Revision 94
Trusted Computing Group. TCG TPM Main Part 3 Commands, July 2007. Specification Version 1.2 Level 2 Revision 103.
Trusted Network Connect Work Group. TCG Trusted Network Connect TNC Architecture for Interoperability, May 2007. Specification Version 1.2 Revision 4.
United States Computer Security Readiness Team. Technical Cyber Security Alert TA08-137A,June 2008.
VMware Incorporated. VMware Virtualization Software, 2008.
Xen Project. The Xen Hypervisor Open Source Project Homepage, 2007.