Trusted virtual Security Module

Design and Implementation

Master's Thesis

at

Graz University of Technology

submitted by

Florian Reimair

Institute for Applied Information Processing and Communications (IAIK),
Graz University of Technology
A-8010 Graz, Austria

January 11, 2011

Advisor: Univ.-Prof. M.Sc. Ph.D. Roderick Bloem

Co-Advisors: Dipl.-Ing. Martin Pirker

Dipl.-Ing. Ronald Tögl

Trusted virtual Security Module

Design und Implementierung

Masterarbeit

an der

Technischen Universität Graz

vorgelegt von

Florian Reimair

Institut für Angewandte Informationsverarbeitung und Kommunikationstechnologie (IAIK),
Technische Universität Graz
A-8010 Graz

January 11, 2011

Diese Arbeit ist in englischer Sprache verfasst.

Begutachter: Univ.-Prof. M.Sc. Ph.D. Roderick Bloem

Betreuer: Dipl.-Ing. Martin Pirker

Dipl.-Ing. Ronald Tögl

Abstract

Cryptography shifts the challenge of securing sensitive data to protecting the used keys. Keys can be tampered easily if unprotected. Security modules are one way of shielding vulnerable keys. They protect a key throughout its lifetime (creation, usage, and deletion) from getting disclosed. Today, low-cost low-security software solutions as well as expensive high-security hardware solutions exist with little in between.

In this thesis we discuss a new type of security module. It operates on commodity hardware and is therefore cheap in cost. A rather high security level is achieved by hardening the core software through technologies like virtualization and trusted computing. A narrow network API offers role-based access to cryptographic services like key management, signature creation and time stamping. Evaluation of our proof-of-concept implementation shows a rather good performance compared to commercial security modules. Technologies like multi-threading and a speed-optimized implementation could boost the performance even more.

All in all, our presented module may be located somewhere between plain software and plain hardware solutions in terms of security and cost.

Keywords: security module, cryptographic services, trusted computing, virtualization

Kurzfassung

Kryptografie verschiebt den Aufwand kritische Daten zu schützen zur Aufgabe die verwendeten Schlüssel zu schützen. Ungeschützte Schlüssel können mit wenig Aufwand angegriffen werden. Security Module sind ein Weg, diese angreifbaren Schlüssel zu schützen. Sie verhindern die Offenlegung der Schlüssel während deren Lebenszyklus (erstellen, verwenden und zerstören). Heute gibt es günstige, unsichere, softwarebasierte Lösungen und teure, hochsichere, hardwarebasierte Lösungen. Dazwischen gibt es wenig.

Wir stellen einen neuen Typ von Security Modulen vor. Er benötigt lediglich handelsübliche Hardware und ist darum günstig. Ein hohes Sicherheitsniveau wird mit Technologien wie Virtualisierung und Trusted Computing erreicht. Eine einfaches Interface bietet rollen-basierten Zugriff auf kryptographische Dienste wie Schlüsselmanagement, einem Signaturservice sowie einem Zeitstempeldienst. Unsere Proof-Of-Concept Implementierung zeigt gute Leistungswerte im Vergleich mit kommerziellen Modulen. Technologien wie Multi-Threading und eine geschwindigkeitsoptimierte Implementierung würden die Leistung noch weiter steigern.

Alles in allem platziert sich unser Modul zwischen reinen Software- und reinen Hardwaremodulen in Hinblick auf Sicherheit und Kosten.

Schlüsselwörter: Security Module, kryptographische Dienste, Trusted Computing, Virtualisierung

Statutory Declaration

I declare that I have authored this thesis independently, that I have not used other than the declared sources / resources, and that I have explicitly marked all material which has been quoted either literally or by content from the used sources.

Eidesstattliche Erklärung

Ich erkläre an Eides statt, dass ich die vorliegende Arbeit selbstständig verfasst, andere als die angegebenen Quellen/Hilfsmittel nicht benutzt, und die den benutzten Quellen wörtlich und inhaltlich entnommene Stellen als solche kenntlich gemacht habe.

Signaturwert	`/Oo3qAXZplRM8LlgTjHJgJ2aA2Yyza3f2mY6uMARNAzTtzm4g2kGVinpZP4yZKG5hOcbijqc3tUy7NwMIQ8p1Q==`	
SIGNATURINFORMATIONEN	Unterzeichner	`serialNumber=819976499486,givenName=Florian,SN=Reimair,CN=Florian Reimair,C=AT`
	Datum/Zeit-UTC	`2011-01-11T09:03:32Z`
	Aussteller-Zertifikat	`CN=a-sign-Premium-Sig-02,OU=a-sign-Premium-Sig-02,O=A-Trust Ges. f. Sicherheitssysteme im elektr. Datenverkehr GmbH,C=AT`
	Serien-Nr.	`470519`
	Methode	`urn:pdfsigfilter:bka.gv.at:text:v1.2.0`
	Parameter	`etsi-moc-1.1@4ebf021d`
Prüfinformation	`Prüfservice: https://www.signaturpruefung.gv.at`	

Contents

List of Figures

List of Tables

1. Introduction

Cryptography shifts the challenge from securing sensitive data to protecting the used keys. Being small and handy, keys can be attacked easily if unprotected. In case of data protected through encryption for example, compromised keys can result in data loss on key destruction as well as in data disclosure on key theft. Compromised hosts, disgruntled staff, and cryptanalysis are listed as the main threats to keys.

Security modules are one way of protecting vulnerable keys against such threats. Basically, a security module protects a key throughout its lifetime. It handles tasks like creation and destruction as well as key usage. An ideal security module never allows key disclosure. In reality, however, a perfectly secure security module does not exist. Two main properties affect the security module market instead: security and cost. Today, there are mainly software security modules (SSMs) and hardware security modules (HSMs) available. Software security modules are in fact a piece of software, be it an application or a library. They need a host platform as well as a host operating system which increases the attack surface. Being low in cost, a software security module provides also a low level of security. Hardware security modules are independent devices that offer a higher security level at higher cost. Therefore, high cost results in a high key protection level, low cost in a low key protection level.

The two main types of security modules leave white spots in product placement (illustrated as question marks in Figure 1.1). Currently, there is no low-cost security module at the market which offers a high security level while providing an adequate performance. The objective

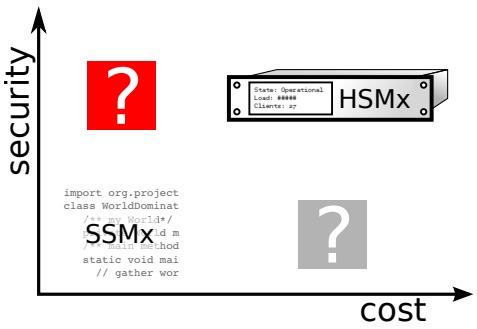


Figure 1.1.: Product placement

of this thesis is to design a security module which fills this gap. The idea is to use trusted computing and virtualization technology to harden a software security module through hardware support and resource isolation respectively. This approach, called the Trusted virtual Security Module (TvSM), promises a low-cost solution with off-the-shelf components offering a rather high level of security. We present a proof-of-concept implementation of our security module.

The basic requirements were creation of a proof-of-concept security module implementation offering key management (creation, deletion, migration, backup), a lightweight Application

Programming Interface, network access, a signature creation service and a time stamping service. Further, the module should withstand common threats. Good performance is an optional requirement. Moreover, a reasonable trade-off should be found between security, cost and usability.

Therefore, our presented design offers a variety of services. Similar to other security modules our design uses a key hierarchy with a so-called master key as a root element. Persistent key storage is done by encrypted export to world readable media. Thus, the module can manage an unlimited number of keys. Key management functionality like key creation, key migration and backup, key import and key export is supported as well. Furthermore, a time stamping service is included. A lightweight network interface [114] allows role-based module operation.

Our solution requires a trusted computing enabled platform which supports Intels TXT. Besides that, any commodity hardware is suitable for hosting the TvSM. A hardened Linux operating system is protected by the Advanced Cryptographic Trusted virtual Security Module (acTvSM) platform [115]. The acTvSM platform provides secure boot, update, and virtualization mechanisms. Given a known-good state value, secure boot ensures that the core executable and therefore the TvSM is reachable if and only if the module is in this state. The module is therefore protected against compromization. The Trusted Platform Module (TPM), which is the hardware part of a trusted computing enabled platform, is used for module identification and master key protection. With its help, a TvSM instance can be limited for use on a specific platform only.

Evaluation of our proof-of-concept implementation shows a rather good performance compared to commercial security modules. Extensive use of multi-threading and a more performance-optimized implementation would boost performance level even more. The security analysis indicates a rather high level of security, which is clearly beyond plain software security modules but also clearly below hardware security modules.

The remainder of the thesis is organized as follows. In Chapter 2 we give background information of our presented security module. This comprises of selected information on cryptography, virtualization, and trusted computing. In Chapter 3 we give an overview over current security modules on the market. The information stated is the result of our market analysis covering about thirty security modules. Chapter 4 we state challenges faced during our TvSM design process. Multiple approaches as well as the selected solution to each challenge are discussed in detail. In Chapter 6 we give results of our comparison of security modules, a discussion on TPM performance, as well as performance evaluation of the TvSM. In Chapter 7 we give details on our proof-of-concept implementation, Chapter 8 concludes the thesis.

2. Background

Prior to designing the TvSM we did a literature research. The goal was to get familiar with the basic building blocks which might be suited for design and implementation of the proposed TvSM: cryptography, virtualization, and trusted computing. This chapter introduces the basic concepts which are necessary for understanding the basic idea, the design decisions made, as well as the inner workings of the proposed TvSM.

First, primitives of cryptography are discussed as well as advanced concepts, including hashing and encryption, as well as authentication and digital signatures. Second, the very basics of operating system virtualization described give an idea on how virtualization supports secure computing. Third, the topic of trusted computing is discussed. Again, all core primitives as well as advanced concepts necessary for understanding the TvSM design are covered. That is roots of trust, sealing, binding, attestation, as well as the architecture of a trusted computing platform, including the Trusted Platform Module and the TCG Software Stack.

2.1. Cryptography

Be it the ancient Egyptians, the ancient Persians or the secret mobile phone call, cryptography has been well used then and now. Preneel says that cryptography is as old as writing itself [102]. Kahn gives a historical overview [88]. So, being the study of information hiding, the original purpose of cryptography was to blind and later unblind information. Today, cryptography is more than that.

In this section primitives of the today's cryptographic toolbox as well as advanced concepts are discussed.

2.1.1. Primitives

Cryptography offers a set of primitives which are used in different applications. In this section we discuss the context-relevant subset of all primitives available. Those are hashing, random number generation, symmetric and asymmetric encryption. At the end of this chapter we give a short summary.

Hashing

For efficiency and data blinding purposes a shorter representation of data blobs, a *fingerprint* or a *digest*, is needed. In cryptography *one-way functions* are used (Figure 2.1). Ideally, these functions should not be invertible. That means, there should be no way to gain information about the input given the result. In practice, limitations show up. The output of a one-way function has to have finite length. Further, the output is defined to be of fixed length which is often shorter than the input. Consequently, those functions are not invertible. But there can be two or more inputs which result in one output value.

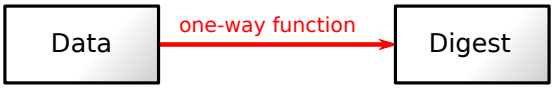


Figure 2.1.: Hashing

For cryptographic use a one-way function has to meet three basic requirements:

- It is hard to find a data blob which results in a given fingerprint. This is referred to as *preimage resistance* in cryptography.
- It is hard to find a second data blob with the same fingerprint as a given data blob. This is referred to as *second preimage resistance*.
- It is hard to find two data blobs which result in the same fingerprint. This is referred to as *collision resistance*.

Given a bit-length n of the resulting fingerprint, cracking the first two requirements needs 2^n fingerprint calculations. A value $n = 64...80$ is believed sufficient to be impractical for computing power available in the near future. In response to the birthday paradox [95], which reduces the complexity of finding pairs to $2^{n/2}$, the length n has to be doubled to meet the collision resistance requirement. In fact, a length of $n = 128...160$ bits and therefore satisfaction of the three requirements turns a one-way function into a so-called cryptographic hash function.

Random Number Generation

An important part of cryptography are random numbers. Random numbers are used whenever a number needs to be hard to guess. A cryptographic key or a number-used-only-once (nonce) are exemplary applications.

An ideal random number is a number which lacks any pattern. That means, the probability of guessing a certain bit of a number's binary representation is 50 percent. The interested reader is referred to [73] for a more complete description.

In practice, there are two design principles for Random Number Generators (RNGs). First, Pseudo-Random Number Generators are a cheap and efficient way of random number generation. They do not really generate fresh random numbers, but use mathematics or lookup tables to produce a deterministic sequence of random looking numbers. Usually, these generators are periodical, which means that a number of a sequence eventually reappears. In practice the period is so huge that the limits are never reached in application. This property makes Pseudo-RNGs a very good and fast source of random numbers in a great range of applications.

Second, True Random Number Generators are commonly more expensive and less efficient than Pseudo-RNGs. They use physical phenomena (mouse movements, network traffic, Nyquist noise, radioactivity, ...) as a source of randomness. Consequently, numbers generated by a True RNG are not deterministic. Each and every number is a fresh random number and is not part of a certain sequence, which does not preclude that a certain number reappears after some time. The downside of True RNGs is their lack of performance compared to Pseudo RNGs. To sum this up, True RNGs are used in applications where non-predictability is more important than efficiency and cost (like cryptography).

Symmetric Encryption

Data blinding is another important part of cryptography. Here, data recovery (decryption) is essential. Generally, there are two functions: one function $c = E(K, p)$ designed for encryption, where p denotes the public readable plaintext, K the shared secret key and c the ciphertext and the other function $p = E^{-1}(K, c)$ designed for decryption (Figure 2.2).

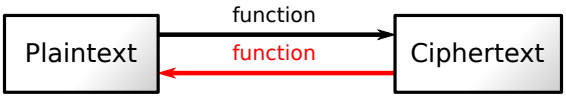


Figure 2.2.: Symmetric Encryption

Decryption of the ciphertext requires the knowledge of a shared key. Consequently, the shared key has to be distributed among all involved individuals. This causes two main problems. First, a secure channel for key distribution is needed to prevent keys from being eavesdropped. Second, one key has to be stored for each individual. The worst case, which is also a common case, is that only two individuals want to share the plaintext. In a static setup avoiding key exchange on demand, a total of $n * (n - 1)/2$ keys are needed in a group of n individuals. That is a total of about 5000 keys given a group of 100 individuals.

Assuming all those challenges can be resolved in a secure manner, symmetric cryptography has one important advantage. In general, algorithms for symmetric encryption are quite fast. Consequently, encryption of large data blobs is possible in a rather efficient way.

Asymmetric Encryption

Asymmetric cryptography, also known as public key cryptography, is a good solution for the key distribution problem faced in symmetric cryptography, among others. A *trapdoor one-way function* [120] is used for this cryptosystem along with a key K consisting of two parts: a world readable part K^{pub} called the *public key* and a part kept secret by the key owner K^{priv} called the *private key*. To encrypt a certain plaintext, the one-way function $c = E(K^{pub}, p)$ is used, where p is the public readable plaintext and c the ciphertext. Decryption is not trivial except on knowledge of the trapdoor, the private key. $p = E^{-1}(K^{priv}, c)$ denotes the decryption process. See Figure 2.3 for a schematic overview.

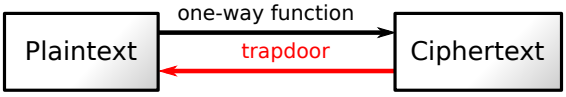


Figure 2.3.: Asymmetric Encryption

Foremost, no secure channels are required for distribution of the public key part. Everyone can use an individual's public key to encrypt a message which is only readable by the individual itself. Performance of public key crypto systems is rather slow compared to symmetric solutions. This is due to the much larger keys required to meet sufficient security levels. Hence, public key crypto systems are often used for the distribution of symmetric keys, whereas bulk data encryption is done by symmetric solutions.

2.1.2. Advanced Concepts

Based on the primitives stated in the previous section, cryptography offers a variety of advanced concepts. In the following, data integrity checks, authentication mechanisms, digital signatures, certification, and public key infrastructures as well as key wrapping are discussed.

Data Integrity

Data integrity checks are meant to prove that data has not been modified. To achieve this, redundancy has to be added to a data blob.

Different methods for different applications are available to create the required redundancy. A simple solution is to duplicate the source data blob and send it twice. If the incoming parts do not match, some modification has occurred. In this case one can state that something went wrong. One cannot say which part was compromised nor can he reconstruct the original message. Further, tampering can be done easily by changing a data blob. Changing the original and all transmitted copies breaks this solution. In communications engineering, for example, error correction is an important point, whereas tamper resistance is insignificant. In cryptography, the main goal is to attest that a certain data blob is unmodified, as well as to make tampering difficult.

This is where hashing (discussed in Section 2.1.1) is used. A hash fingerprint is constant and small in size and a small change of the input data results in a major change of the fingerprint. This allows verification of a data blobs integrity. It works as follows: a cryptographic hash $h = H(p)$, where $H(\cdot)$ denotes a cryptographic hash function and p the plain data blob, is calculated by an individual, for example Alice. The tuple (p, h) is transfered to another individual, Bob, for example. Bob takes the plaintext p and derives the cryptographic hash $H(p)$. If the values h and $H(p)$ match, there was no modification. Of course, a plain hash is exposed to tampering without further protection. One can take the data blob, modify it, recalculate the fingerprint and forward the compromised tuple to its intended destination.

With a shared secret K, Message Authentication Codes (MACs) fix the vulnerability of plain fingerprints. A MAC is computed by $MAC = H_K(p)$. The result therefore depends on the secret K which is not known by a third party. Concatenation of the public readable data and the secret key gives a simple realization: $MAC = H(p||K)$ where $\cdot||\cdot$ denotes bit-level concatenation. Like symmetric encryption this solution suffers from the key distribution problem.

Authentication

Whenever it is important to know who is acting, cryptographic authentication is needed. The goal is to prove that one particular entity has done something and nobody else.

A shared secret can be used for individual authentication. It works as follows: Alice and Bob share a secret authentication key. If Bob somehow includes this shared key in a message, the receiving part, Alice, can verify that this message is from Bob. Like symmetric encryption (discussed in Section 2.1.1) this solution needs a total amount of $n * (n-1)/2$ keys for a group of n individuals to perform authentication. Therefore, this is not used in practice.

Asymmetric encryption can also be used for individual authentication. It works very similar to the asymmetric encryption scheme discussed in Section 2.1.1 with the difference that Alice

uses her private key for encryption. Therefore, everyone is capable of message decryption. By successful decryption of the message with the public key of Alice it is shown that the private key of Alice was used for encryption. Alice is the only one capable of her private key. Consequently, Alice has encrypted the message and no one else. This scheme has similar advantages as the asymmetric encryption scheme discussed in Section 2.1.1.

In practice, digital signatures are used for message authentication. They are more robust against attacks than the methods discussed yet.

Digital Signatures

Digital signatures are meant to bind a certain data blob, a message for example, to a certain person. The technique of digital signature creation and verification uses data integrity (discussed in Section 2.1.1) and asymmetric authentication mechanisms (discussed in Section 2.1.2) to gain a robust tool. The goal is to provide data integrity, non-repudiation and authentication at once and excluding the secured data exchange procedure. Therefore, digital signatures can be seen as the electronic equivalent of manual signatures on documents [102].

One simple realization works as follows: Alice hashes her plaintext p and encrypts the result with her private key $s = E(K^*, H(p))$ where s denotes the resulting signature value, $E(K^{priv}, \cdot)$ the encryption function with the private key K^{priv} and $H(\cdot)$ a cryptographic hash function. Now Alice sends the tuple (p, s) to Bob. Bob decrypts the signature value s with the public key of Alice K^{pub} which results in the hash value provided by Alice. Signature verification is done by comparing the hash of the plaintext with the decrypted hash provided by Alice: $H(p) \stackrel{?}{=} E^{-1}(K^{pub}, s)$, where $E^{-1}(K^{pub}, \cdot)$ denotes the decryption function using the public key K^{pub}. If the comparison holds there are a couple of facts one can state. First, the message is unchanged due to matching hashes. Second, the message was signed by Alice. Only Alice knows her private key and decryption using the public key of Alice results in a valid hash value. Third, additionally Alice cannot refute or repudiate the fact that she has signed this document.

Other signature creation schemes are not directly derived from the public key encryption scheme like [54], [84] or [93]. They use other mechanisms to provide digital signature creation and verification. This is not important for understanding the topic of this work. Therefore, further reading is left to the interested reader.

Public Key Infrastructures

The basic parts of a public key infrastructure (PKI) are digital certificates. A certificate contains details about a certain person, including its public key data, among others. Further, the validity period of the certificate itself, the certificate issuer and other information are included. The whole document is signed by a Certification Authority (CA), which is the second important part of a PKI. A Certification Authority is a trusted party with signing abilities. Whenever a certificate gets signed by a CA it can be used to sign other certificates. Consequently, a chain of signatures is formed which originates in the very first root certificate. There is no way to attest trust for this root certificate. Therefore, a known-good trusted third party has to be used. See Figure 2.4 for an illustration.

A PKIs is an environment to manage certificates. Certificates can contain all kind of public key data. Consequently, certificates can be used for digital signature creation as well as for

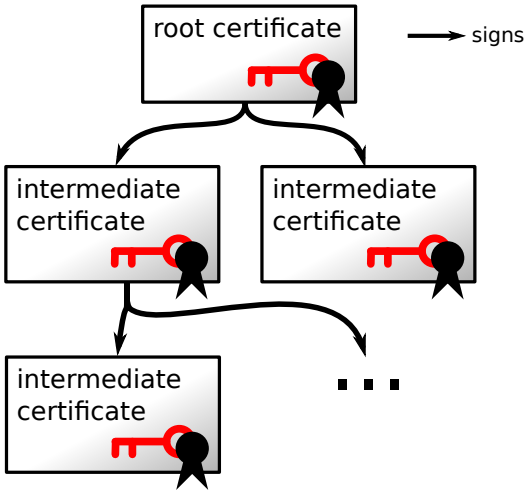


Figure 2.4.: Exemplary certificate hierarchy of a public key infrastructure

data encryption tasks.

Wrapping

Key wrapping is a key storage strategy. Its goal is to provide secure key storage on public readable media. See Figure 2.5 for an illustration.

A cryptographic key K has at least a private part K^{priv}. Keys for asymmetric cryptography also have a public part K^{pub}. The goal is to protect the private part of a key from eavesdropping. The wrapping strategy uses one (private) key to encrypt other (private) keys. In this context an encrypted key is called a wrapped key. A wrapped key is neither readable nor usable without first being decrypted by its parent.

$$wrap\ K_1 = (K_1^{pub}, K_1^{priv})\ with\ K_2 \leftrightarrow (K_1^{pub}, E(K_2^{priv}, K_1^{priv}))$$

with $E(x, \cdot)$ being the encryption function using the key x. This, again, forms a chain originating in the very first key, the so-called root key. The root key cannot be protected with use of key wrapping. Therefore, other mechanisms have to be used. Each child of the root key is protected by its parent and therefore secure as long as the root key is secure.

2.1.3. Summary

In this section a set of cryptographic primitives available was discussed. First, hashing as a tool to get a short and constant in length *fingerprints* representing data blobs was covered. Then, the basis of every cryptographic key, Random Number Generators were discussed as well as encryption as revertible operation of scrambling a data blob. Furthermore, concepts which make use of the stated primitives were covered. Ensuring data integrity, authentication and digital signatures, as well as public key infrastructures and key wrapping was discussed.

The methods reflected in this section are the basics necessary for understanding the idea behind the TvSM, the design decisions made and the inner workings as well.

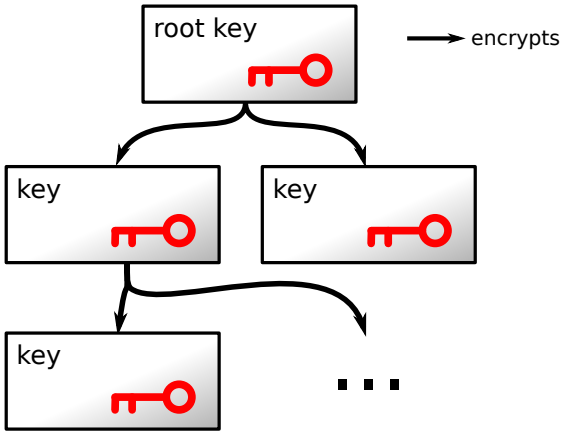


Figure 2.5.: Key Hierarchy

2.2. Virtualization

When it comes to virtualization there are different points of view. In industry the first question usually is about the economic advantages of virtualization. Server administrators ask for technical characteristics. A third important point is security. Virtualization claims ease of usage, flexibility and reliability. Also, resource maximization as well as reduced business continuity costs are mentioned. An overview is given by [86] and [87].

2.2.1. Advantages

Virtualization can be seen as an additional abstraction layer between a platform's hardware and the operating system. This has several advantages. An operating system does not have to communicate with the hardware directly. Therefore, no individual drivers have to be used. This makes portability easy. Virtualization makes it possible to just move an entire operating system to another platform without changing the operating system.

The abstraction layer also makes it possible to run more than one operating system on one physical platform. An operating system running on a virtualized platform among others is called guest and runs in so-called compartments. Guests are separated from each other in terms of file systems, memory and other resources. Furthermore, resources used by each guest can be controlled individually allowing specific load balancing constraints. From an economic point of view cost savings might be possible by server consolidation. Different operating systems fulfilling specific tasks on separated server machines can be moved to a virtualized platform as guests. Thus, there is no need for dedicated hardware for every single service anymore. Less hardware leads to less components where failures can occur. With this, the chance of hardware failure is reduced as well. If a failure occurs it is more critical because it affects multiple services.

Nonetheless, administrative tasks such as setting up operating systems on new servers can be done more efficiently. All one needs to do is create one basic operating system installation. The resulting hard disk image can be copied to a new compartment of a virtualized platform and booted up as a new clean operating system. This is faster than a traditional setup on real hardware in most cases. From an economic point of view this results in shorter setup times.

In case of a hardware failure (given a running backup machine) or hardware upgrade handling one physical platform can be done faster compared to handling a couple of different platforms. Therefore, downtimes decrease and upgrading requires less hardware.

Finally, security policies can be defined more efficiently. In theory, the compartments of a virtual machine are strictly separated. Consequently, each guest can rely upon the fact that its resources are not accessible by any other guest. This eases administration tasks such as access restrictions, write/read permissions or license management as well as service level agreements.

2.2.2. Virtualization techniques

Since the early beginnings of hardware virtualization (1970s [85]) a variety of techniques arose. One technique is called operating system-level virtualization (Figure 2.6a). One single operating system provides several user-spaces with isolated file systems and resources. This is a minor upgrade to common operating systems. Linux-VServer [17] as well as OpenVZ [18] implement this virtualization technique.

The requirement after a more strict resource isolation lead to a more sophisticated technique called para-virtualization [118] (Figure 2.6b). This technique utilizes a so-called Virtual Machine Monitor (VMM) or hypervisor. The hypervisor, itself a piece of software, mediates between the guest operating systems and the underlying hardware. Guests have to be modified to fit into this virtualization technique. A common product implementing this kind of virtualization is the first version of Xen [66].

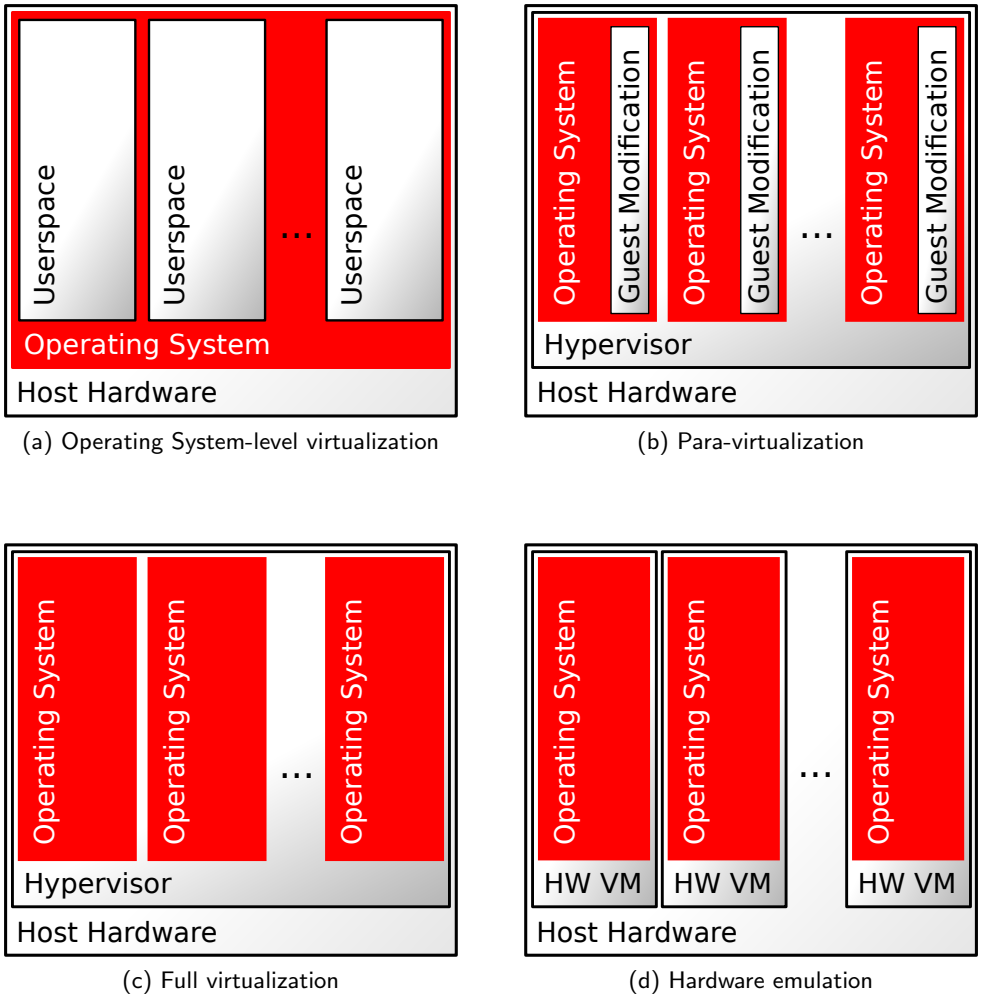
VMWare ESX/ESXi [14] as well as z/VM [58] and others accomplish the virtualization task in a similar manner as para-virtualization does, but without guest modification. This technique is called full virtualization [118] (Figure 2.6c). Binary translation techniques are used to virtualize the underlying platform. Often, this solution lies behind para-virtualized solutions in terms of performance.

A new virtualization technique was made possible by recent advances in Central Processing Unit (CPU) development like Intels Virtualization Technology (VT-x) [99] and AMDs Virtualization Technology AMD-V [2]. Called hardware assisted virtualization technology [118] modern CPUs offer enhanced instruction sets to assist transparent virtualization services. Again, a hypervisor is needed for mediating guests (Figure 2.6c). Newer versions of XEN [109] and KVM [90], among others use this new technology.

Finally, there is another technique called hardware emulation (Figure 2.6d). Software models of real hardware devices are build and offered for usage. This is a complex task and therefore performance is rather slow. Common hardware emulators are Bochs [15] as well as QEMU [19] or DosBox [16]. They are often used to emulate certain hardware devices on common desktop computers for development purpose.

2.2.3. Summary

All in all, virtualization offers a tool which reduces the need for hardware, which reduces setup-times as well as down-times due to hardware failure/upgrade. Further, it preserves resource isolation and simplifies management tasks. Different virtualization techniques, hardware virtualization, para-virtualization, full virtualization, and hardware assisted virtualization, have been implemented by different vendors/projects. All in all, virtualization promises a high security level due to its isolation capabilities, which makes it suitable for the TvSM design.



(a) Operating System-level virtualization

(b) Para-virtualization

(c) Full virtualization

(d) Hardware emulation

Figure 2.6.: Virtualization techniques

2.3. Trusted Computing

Trusted computing is a rather new technology driven by the Trusted Computing Group (TCG) [13]. Its goal is secure personal computing. Unlike the usual procedure of finding an fixing bugs which allow an attacker access to a system trusted computing uses cryptography to measure a systems state. The system's state depends on the software components which are executed as well as on the sequence of execution. Any state change indicates a change in software setup. A change in software setup can be triggered by a regular software update as well as by a piece of injected malicious code. The source of the system state change has not to be known. In other words, trusted computing tries to secure a trusted computing-enabled system, named trusted computing platform (TCP), against yet unknown attacks which may arise in the future.

In this section the primitives of trusted computing as well as advanced concepts are summarized. Tools like measurement and authorization are described. Advanced services like attestation, binding, and sealing are covered as well. The common architecture of a platform

capable of trusted computing followed by drawbacks and open issues close the section.

2.3.1. Tools

There is a variety of tools available within a trusted computing platform. These tools are the basis for the services discussed later.

Chain of Trust, Roots of Trust

A chain of trust (CoT) is used to extend trust over a whole system. It works as follows. A known-good and therefore trusted part verifies a yet untrusted part. On success, trust is extended to the new part. Consequently, the new part is now known-good and trusted. It can verify other parts of the system and so on. Logically, the very first part of a chain of trust cannot be verified. This is where roots of trust come into play.

A Root of Trust (RoT) is a hardware or software mechanism that one implicitly trusts [83]. Consequently, a chain of trust can only originate from a Root of Trust. Clearly, the whole chain of trust and therefore the whole system relies on its RoT. In case of a faulty RoT (compromised, erroneous, ...) the whole system cannot be trusted.

The TCG defined three roots of trust. The Root of Trust for Measurement (RTM), the Root of Trust for Storage (RTS) and the Root of Trust for Reporting (RTR). Together they form the Trusted Building Block within the specification of the TCG [98, p. 27]. In a trusted computing platform the RTS as well as the RTR are realized by the TPM (see Section 2.3.3). An example for the RTM is the systems Basic Input Output System (BIOS).

Measurement

To build a chain of trust some sort of measurement m has to be taken. In case of a trusted computing platform, objects to be measured are binary images, executables, libraries and so on. The TCG a secure hash algorithm to obtain measurements of objects [39]. Other solutions might be possible and are object of ongoing research [104]. These measurements have to be protected from tampering to preserve integrity and trust. In a trusted computing platform the TPM serves as secure storage.

Furthermore, the TCG also defines sequence to be important. There are two main reasons for that. First, if one can show that a chain of trust was build in the same sequence as before, the measurement is more reliable than without sequence. Second, a much more efficient way of realization is possible. The TCG defines the so-called *extend operation* as

$$m_t = H(m_{t-1}||H(object)),$$

where t denotes the state, $\cdot||\cdot$ bit-level concatenation, $object$ denotes the current block of data to be measured and $H(\cdot)$ is a cryptographic hash function. In words, each standalone measurement $H(object)$ is concatenated with its predecessor m_{t-1} and hashed again to form the new measurement m_t. That way, the sequence of measurement and the measured objects are included in one single value m.

A trusted computing platforms secure storage, the TPM, offers a set of shielded storage slots for measurement values. In the TPM specification they are referred to as Platform Configuration Registers (PCRs). More PCRs allow more than one concurrent chain of trust.

This feature allows CoT branching as well as independent CoTs for different user applications for example.

Authorization

Accessing items stored inside the TPM such as data or keys as well as usage of certain services such as key generation or key migration requires some sort of authorization. The TCG defines identity-based authorization as showing the knowledge of some sort of secret value. The common case being passwords used as secrets.

The TCG defines three different privilege levels. The owner-mode, the user-mode and the no-authentication mode. The owner is identified via his secret. This secret is inserted during the take-ownership procedure where an individual takes ownership of a TPM. Taking ownership of the TPM implies taking ownership of the trusted computing platform where the TPM is built-in. While the owner can run privileged operations such as managing access constraints on storage locations or resetting the TPM, he does not have access to the whole TPM with all its stored data. A standard user does not need any authentication data to use the services of the TPM. He can generate keys and use them as long as the required key secrets can be provided. The no-authentication mode allows low security operations such as reading the TPMs version.

2.3.2. Services

Based on the tools discussed before a trusted computing platform offers a variety of services to enhance computational security.

Attestation

Adducing evidence of a system being trustworthy is called attestation. Three steps are necessary to perform this task. First, the boot chain has to be measured. Second, the measurement result has to be reported. Third, the result is left to be verified remotely.

Measurement is done with help of the measurement capability (Section 2.3.1) of a trusted computing platform. Data and sequence are tracked by the TPM. Figure 2.7 shows a trusted boot process initiated by the Root of Trust for Measurement (RTM). Each new object is measured before control is handed over to it. Thus, a chain of trust is established.

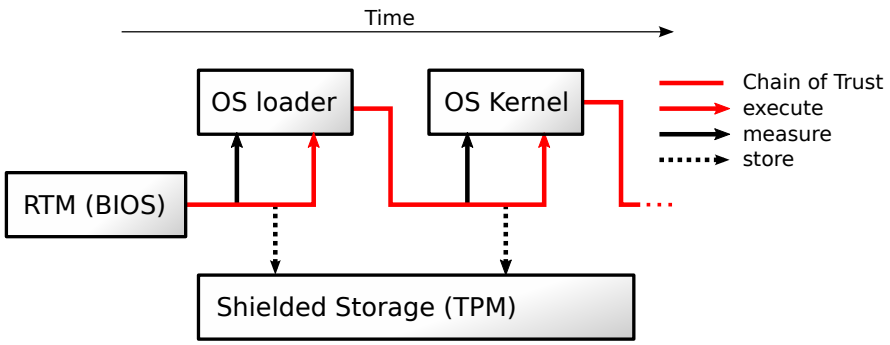


Figure 2.7.: Boot process resulting in chain of trust

To securely report the result to a remote host, cryptographic techniques, namely digital signatures (see Section 2.1.2), are used. Each TPM has a unique Endorsement Key (EK). The private part of this asymmetric key pair never leaves the TPM. So called Attestation Identity Keys (AIKs) are used as aliases for the TPMs EK. The TPM signs the result with one of its AIKs to certify the reported value. Thus, there is no need for a secure channel between the TPM and the remote host. Consequently, secure reporting can be done via the Internet for example.

Given the result of measuring the boot chain now an informed decision can be made whether or not the system is trustworthy (secure). Verification is done against a known good value. There are different ways to get a known good value. One can take the result of the last boot. If the values match, evidence is given that the system has not changed since the last boot. In case the system was trustworthy then so is it now.

On platforms like mobile phones, PDAs etc. another solution is more common. Each software version released by the vendor ships with a certified known-good state value. Now every time the system boots up the resulting measurement is compared to the certified value of the vendor. In case of a match the systems trustworthiness is shown (secure boot). The challenge remaining is how to handle software updates (see Section 2.3.4).

Binding/Sealing

Further services provided by a trusted computing platform are data binding and data sealing. Binding reflects the process of encrypting data with a key maintained by the TPM. A private key never leaves the TPM unencrypted. Therefore, the data is bound to the TPM, because only this TPM can decrypt it. Hardware limits the amount of data which can be handled. It is limited by the size of the used key as well as the used padding scheme. For example, with an RSA key size of 1024 bits used with Optimal Asymmetric Encryption Padding (OAEP) a total of 85 bytes (= 680 bits) can be encrypted [74, p. 129]. Given this constraint, although symmetric cryptography is not supported natively by the TPM, symmetric keys can be managed in terms of creation (Random Number Generator) and encrypted storage.

Data sealing enhances data binding. It provides a way to combine the systems state and data binding. Additionally, the data is restricted to a given platform state. In other words, a data blob can only be decrypted if the key data and the system state are as they were set at the point of encryption.

2.3.3. Architecture of a trusted computing platform

The architecture defined by the TCG is a layer-based design. In theory, each layer can be exchanged without touching adjacent layers. See Figure 2.8 for an overview.

The basis of every system is some kind of platform. In a trusted computing platform the platform includes a TPM as well as a Root of Trust for Measurement (RTM). On top of the platform there is the operating system which manages resources, provides hardware drivers and so on. The hardware driver of the TPM is called TPM device driver (TDD) and operates in privileged mode. It is provided by the vendor of the TPM. Besides the TPM the TCG Software Stack (TSS) is the second important part of a trusted computing platform (highlighted red in Figure 2.8). The TSS runs as unprivileged system process inside the operating system. It is composed of three logical parts: the TCG device driver library (TDDL), the TCG core

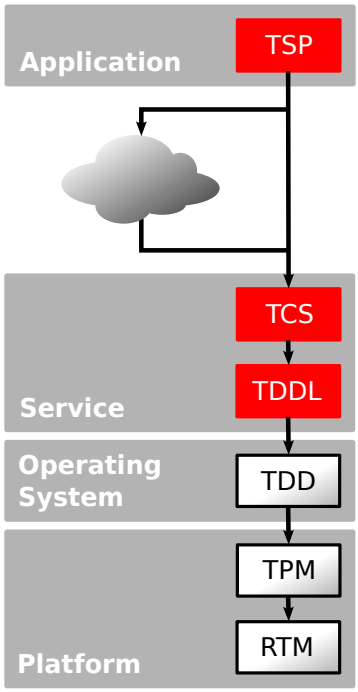


Figure 2.8.: Architectural overview of a trusted computing platform

service (TCS), and the user library TCG service provider (TSP) [74, p. 77 ff.]. In this section the individual parts are explained in detail.

Trusted Platform Module

In order to create a trusted computing platform, a dedicated hardware security module called Trusted Platform Module (TPM) is included in the specification of the TCG. The TPM is very similar to a SmartCard (SC). It provides cryptographic services, storage, key management as well as other protected capabilities accessible via a well defined interface. Furthermore, tamper resistance against software attacks is implemented. The main difference to SmartCards is that the TPM is physically bound to its host platform. Consequently, the TPM can identify the uniqueness of its platform with its Endorsement Key (EK). That allows operations such as identification or authentication. Drawbacks such as privacy issues arise as well. Furthermore, TPMs should be devices which do not have great cost impact on host systems. On the one hand, this helps spreading trust and trusted computing (TC) to every digital device. On the other hand, resources offered by the TPM such as speed and memory remain small and therefore only a small set of services is feasible.

Cryptographic Services As an HSM, the TPM offers a variety of cryptographic services. First of all a hardware Random Number Generator (RNG) is included within the chip. This service is public usable and guarantees high quality generation of keys and other random data based on entropy provided by the systems hardware such as thermal noise (Johnson-Nyquist noise) [98, sec. 4.3.1.3].

Furthermore, symmetric cryptography is included in the TPMs specification. The symmetric cryptographic engine is only used by the TPM itself. Examples are protection of user authentication data or securing communication sessions. The Vernam scheme [117] with 160 bits in length and use of the XOR-operation to encode the data with the key is the minimal requirement stated by the TCG. More advanced techniques such as the Advanced Encryption Standard (AES) can be used as well [98, sec. 4.3.1.1].

Another service, which is only available inside the TPM is the keyed-Hash Message Authentication Code (HMAC) engine. It is used for securing communication from the user (TSP for example) to the TPM in terms of integrity checking and authorization.

The Secure Hash Algorithm (SHA)-1 is used for secure hashing. The service is used internally by the HMAC engine as well as for extending measurements in the context of trusted boot (see Section 2.3.1 and Section 2.3.2). Furthermore, the SHA-1 engine can be used externally without authentication. Due to the limited resources of the TPMs and the limited throughput of the communication line, the engine is rather slow. External software solutions are preferred, if available.

For asymmetric cryptography the well known RSA public key crypto system was choosen by the TCG. Key sizes of 512, 768, 1024 and 2048 bit are required by the TCG specification version 1.2 [39]. The engine is used for encryption/decryption as well as for digital signatures.

The last service discussed here is time stamping. Due to cost reasons the TCG decided that no trusted Real Time Clock (RTC) has to be implemented into TPMs. As an alternative, another service called tick counter was included, which is in fact a subset of a secure RTC. A random number is generated every time the TPM is powered up. The internal tick counter is initialized with this random number. The counter is incremented every tick, which is fired by an internal oscillator. This allows the TPM to keep track of the amount of time passed since startup. For reporting, the initial value as well as the current counter value are put into a data blob and signed by the TPM. Given the accuracy and precision of the TPMs internal clock, the tick count can be associated to a known period of time. Therefore, the values can be used to create a real time stamp utilizing a Time Stamping Authority (TSA) or just a trusted report on how much time has gone by. [74, p. 276][89, p. 295]

Authorization Almost all operations offered by the TPM require authorization. As already mentioned in Section 2.3.1 the TCG defined identity-based authentication using secrets such as passwords or Personal Identification Numbers (PINs). Each object managed by the TPM can be bound to a secret. To use a protected object, the corresponding secret has to be provided. Furthermore, a secure channel between client and TPM has to be established to ensure secure communication. The TCG defined a collection of authorization protocols. The most important are Object-Independend Authorization Protocol (OIAP) and Object-Specific Authorization Protocol (OSAP). All protocols use the *rolling nonce paradigm*.

The *rolling nonce paradigm* ensures prevention from replay attacks as well as man-in-the-middle attacks at the outset [40, Chap. 13]. A nonce is created by the requesting peer of the communication line and sent to another peer as part of the request. The other peer includes the received nonce in its response to the requester. Thus, the initiating peer can validate the nonce. After successful validation of the response a new nonce is created for a new request.

The Object-Independend Authorization Protocol is not bound to any TPM object. After an initial session key exchange the session is opened for all kind of operations on almost all

objects managed by the TPM. The session has to be closed explicitly by a subscriber. [98, Chap. 4.3.2.1]

The Object-Specific Authorization Protocol, on the contrary, is used to authorize just one specific TPM object such as keys or data blobs. As long as the session is open different operations can be performed on/using the authorized object. Furthermore, OSAP also offers line encryption as an option to scramble communication.

Further, the TCG specification defines the Delegate-Specific Authorization Protocol (DSAP), the AuthData Insertion Protocol (ADIP), the AuthData Change Protocol (ADCP), and the Asymmetric Authorization Change Protocol (AACP) for use with the TPM [40, Chap. 13].

Key Management One of the core concepts of trusted computing is asymmetric key management. Keys can be created, imported, stored, provided with usage constraints, protected with secrets, wrapped and migrated. The Storage Root Key (SRK) implements the Root of Trust for Storage (RTS) (see Section 2.3.1).

A new key can be either created with help of the internal RNG or imported from an external source (constraints are given by the migration choice. this will be discussed later). Moreover, an existing parent key has to be specified for key wrapping (see Section 2.1.2 for details on key wrapping). Wrapped export features archival storage of keys on external mass storage devices. Further, a key hierarchy is obtained. Figure 2.9 shows a sample configuration.

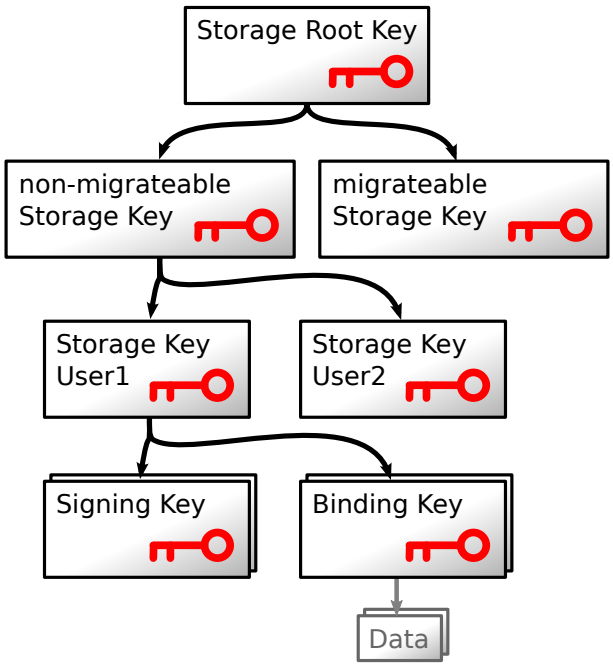


Figure 2.9.: Exemplary key hierarchy

A key secret can be assigned to a newly created key. Consequently, only the one who knows the secret is able to use the key. Even the privileged TPM owner (the system administrator for example) does not have access to the key for example.

Furthermore, a key can be marked migratable or not. Key migration is a technique to move/copy keys from one TPM to another. This can be necessary for backup reasons, or, if

one key is required on two platforms. If a key is tagged migratable a so-called migration secret has to be selected. Knowledge of this secret is required to migrate a certain key. Furthermore, only the TPM owner can choose the target parent key (from another TPM). Therefore, migration is only possible if the owner of the TPM as well as the owner of the key consent to migrate the key. [74, p. 34 ff.]

Also, the use of keys can be limited to certain services like storage, binding, signing and identification. Storage keys are privileged to wrap other keys. Thus, those keys are used to form a certain hierarchy as illustrated in Figure 2.9. Their size is set to 2048 bit in prior to provide the highest possible security in terms of the state-of-the-art TPMs. Binding keys are meant to protect a data blob of a given maximum size (see Section 2.3.2 for further details). In most cases, a symmetric key which is used for bulk encryption is bound. Signature keys are restricted for signature creation use only. The RSA signature scheme is used with no size constraints. Attestation Identity Keys (AIKs) are always non-migratable keys and have the SRK as parent. They are created inside the TPM and certified by a trusted third party, which links the certificate to the TPMs EK. Therefore, they can be used as alias for the TPMs EK without revealing it. Consequently, AIKs represent one the TPM's identity.

TCG Software Stack

As mentioned previously, the TSS is a userspace system process providing an Application Programming Interface (API) to the TPM. Its three logical parts are described in the following paragraphs.

TCG device driver library (TDDL) The TDDL provides an API to communicate with the TPM device driver. Basic commands such as send and receive data blobs and a few more are provided by this driver library. For resource constraint trusted computing platforms it is possible to use this library as interface to the services of the TPM. Typically, the TCS is the user of the TDDL.

TCG core service (TCS) The TCS is the core part of the TSS. It provides all kind of services to use the underlying hardware TPM. Furthermore, the TCS is limited to one instance per TPM. Additionally, the API offers advanced services such as a command blob generator, authorization sessions, key management and application access synchronization to ease up usage of the TPMs resources.

TCG service provider (TSP) The TSP is the API which is used directly by an application. Its intention is to close the gap between TCS and the user application. It is realized as a dynamically linked library to offer easy access to TC. It can be used directly by the application or by a remote platform utilizing a protocol such as Simple Object Access Protocol (SOAP).

2.3.4. Drawbacks and open issues

Although many security improvements are claimed by the Trusted Computing Group, there are also drawbacks and open issues. An overview is given in [94].

One point is that the unique identity of a TPM, given by its Endorsement Key (EK), may result in loss of privacy. There are already two solutions provided. The first solution is the use of Attestation Identity Keys (AIKs) derived from a certain EK. This requires an online connection to a highly available trusted third party. Given that, a key can be certified and keys in question and their certificates can be checked. The second solution is called Direct Anonymouse Attestation (DAA) [71]. This solution works without a trusted third party. In practice the required calculations are too complex to be handled on portable devices such as mobile phones or SmartCards [80].

Another issue concerns the trusted boot process (see Section 2.3.2 as well as Section 2.3.1). Due to the fact that the sequence of objects within the boot process affects the final result of measurement, a challenge arises on today's personal computers. The ability of real parallelism, provided by hyperthread-enabled multicore CPUs causes a unknown entropy in the sequence of a systems boot process.

Further, no practical solution was found yet to software changes caused by updates and bug fixes. A very similar situation occurs in handling of configuration files. Given a sealed data blob, it would not be decryptable anymore due to the changed measurements.

2.3.5. Third Party Extensions

Other technologies extend the TCG Trusted Computing design. In the following, three important solutions namely Intel Trusted eXecution Technology (TXT), AMD Presidio Technology and IBM SecureBlue will be discussed.

Intel Trusted eXecution Technology The Intel Trusted eXecution Technology (TXT) is one of Intels contributions to trusted computing. An architectural overview is given in [32]. Beside usage of a TPM the technology enhances several hardware components such as CPU, chipset, memory controller and input/output (I/O) to meet the requirements of a trusted computing platform.

The *late launch* procedure of Intels TXT allows launching a measured Virtual Machine Monitor (VMM) at runtime [83, Chap. 11]. That is achieved by stopping all processing units except one. The active CPU then executes a certified component, a binary representing a dynamic Root of Trust, in a shielded memory of the processor. Tamper detection mechanisms allow the binary to initiate a chain of trust by extending its trust to another (software) component. Now, all processing units can proceed with their operations. Given a chain of trust, a trusted VMM can be loaded for instance. It can be used to maintain compartments in terms of creating, booting, and deleting.

AMD Presidio Technology AMD's Presidio is similar to Intels TXT. Again, protected execution capability is offered. AMDs solution, for example, supports but does not provide a certified component which then triggers creation of a chain of trust. Further, no dedicated cache environment is available for execution of a certified component. [98, p. 68 ff.][110]

IBM SecureBlue In 2006, IBM presented *SecureBlue* [116]. Preceding publications had been made in 2001 [96] but nothing has followed yet.

SecureBlue addresses the low performance of the low-cost TPMs. The goal is to include hardware cryptographic engines directly into the CPU. That promises a performance boost for use of various cryptographic services as well as improved security due to direct communication with the TPM [98, p. 69]. Initially designed for IBMs PowerPC architecture *SecureBlue* will be integrateable in various general purpose processors from Intel and AMD.

Summary

The extensions made to standard platforms try to fix drawbacks of the TCG design. Intel and AMD offer protected execution environment ability, IBM will offer cryptographic enhancements for CPUs to support low performance TPMs. All in all, these technologies add well to the TCG specifications and offer new fields of application for trusted computing in general.

2.3.6. Summary

In this section the basic idea as well as the basic techniques of trusted computing were covered. Primitives like measurement and authorization as well as advanced concepts like attestation, binding and sealing were discussed. Further, a typical trusted computing platform and its parts were discussed. Finally, open issues as well as supporting technologies were stated.

Again, the contents covered in this section are filtered to support the understanding of the TvSM idea as well as its design. Further information can be found at the TCG website [13] as well as in relevant literature.

3. Security Modules

To get familiar with the topic of security modules we did a market analysis with about thirty modules. The list can be found in Appendix A Table A.1. The goal was to get an overview of features and services provided by off-the-shelf security modules. Besides the product briefs, no technical documentation of the reviewed modules was available from the product vendors. Nonetheless, we found test reports for standardization as an excellent source of information. Given that, we evaluated the most important standards concerning security modules first.

The chapter is organized as follows. First, we summarize and discusse the most important standards which we reviewed. Second, we state fields of application for security modules. Next, we give an overview of provided cryptographic services followed by key management. Further, we cover communication types and authentication as well. Backup and disaster recovery closes the chapter.

3.1. Standards

Standards provide well defined methods and tools to evaluate certain products. Therefore, products can be compared in a impartial manner. Furthermore, if a product is certified, it was reviewed and tested by a third party for meeting the given requirements. In the area of security this is very important.

In the following we discuss the standards ISO/IEC 15408, FIPS 140, and ISO/IEC 19790.

3.1.1. ISO/IEC 15408

Named Common Criteria for Information Technology Security Evaluation (Common Criteria), the ISO/IEC 15408 standard was made to provide certainty that a security product, a security module for example, meets the standards requirements and therefore provides a certain level of security. The standard was enhanced and refined several times. Currently the standard consists of three parts. The first part was initially published in 2005 [27] and refined in 2009 [55]. The remaining parts were published in 2008 [42, 43].

The key concept is to validate claims about a certain product. The process is done in two steps. In the first step, the subject of the evaluation, the Target of Evaluation, is defined by selection and/or creation of a Protection Profile and a Security Target including Security Functional Requirements. A Protection Profile describes a certain group of devices. It is done by a user community, a group of developers or by governments and large cooperations. Protection Profiles can be used as evaluation targets or as templates for a product Security Target. Therefore, a customer can select an appropriate Protection Profile and compare products validated against it. A Security Target is a list of selected requirements (Security Functional Requirements) for one particular product.

Table 3.1.: Evaluation Assurance Levels defined by Common Criteria [43]

code	in words
EAL 1	functionally tested
EAL 2	structurally tested
EAL 3	methodically tested and checked
EAL 4	methodically designed, tested and reviewed
EAL 5	semiformally designed and tested
EAL 6	semiformally verified design and tested
EAL 7	formally verified design and tested

The second step is the verification step. It uses the Protection Profile and the Security Target as input. A subset of the Security Assurance Requirements provided by the Common Criteria is selected for the actual evaluation step. The available Security Assurance Requirements cover a simple functional test and methodical checks as well as semiformal and formal verification.

Finally, evaluation results in a certain Evaluation Assurance Level (EAL). Common Criteria defines seven different levels, EAL 1 to EAL 7 (see Table 3.1). Consequently, a customer can decide which level is appropriate for his requirements.

3.1.2. FIPS 140

The Federal Information Processing Standardization (FIPS) 140-1 standard [21], issued on 11 January 1994, was the first standard within Series 140 of the National Institute of Standards and Technology (NIST). Developed by a working group consisting of representatives of the government of the United States of America as well as representatives of the industry, requirements and methods were created to rate certain security modules. A new version, FIPS 140-2 [23], adjusted to state-of-the-art technology and enhanced based on feedback from various institutions such as testers and vendors was released on 25 May 2001. A third version, FIPS 140-3, is currently developed. On 11 December 2009 the revisited draft of the standard was published [60].

Eleven different areas are covered by the standard. Ports and Interfaces, Roles, Authentication, Physical Security, Operational Environment, Cryptographic Key Management and a couple more are the areas which are to be evaluated.

Certified security modules are approved to one out of four security levels. Level 1 is the lowest level of security available within the standard. A security module approved to level one has to meet basic security requirements such as use of at least one approved algorithm or one approved security function. No physical security mechanisms are required. Hence, this level is the only level software security modules can be approved to.

For a level 2 certification, additional requirements have to be met. First, role-based authentication has to be offered. Each role allows a specific set of services. Common roles are a user role and an administration role. Second, unauthorized access to the modules crit-

ical security parameters has to be prohibited by use of tamper-evident covers, doors and/or locks. Further, a modules software/firmware can be executed on a general purpose computing system whose operating system meets functional requirements in the Common Criteria Protection Profile (PP)s as well as is evaluated at the Common Criteria EAL two or higher (see Section 3.1.1). Moreover, a trusted operating system may be used which is evaluated in an equivalent way.

Requirements of level 3 extend those given in level two. First, identity-based authentication has to be offered. An approved security module decides upon the identity of an operator which role he is authorized for. Second, tamper-evidence has to be enhanced by intrusion detection mechanisms. These mechanisms are intended to trigger zeroization of all plaintext critical security parameters on unauthorized access. Third, I/O of critical security parameters in plaintext requires dedicated ports which are physically separated from other ports. A logical separation is allowed also if some sort of trusted path is used. Beside that, only wrapped I/O of critical security parameters is allowed (see Section 2.1.2 for more information about wrapping). Further, the module's software/firmware can be executed on a general purpose computing system only if the used operating system meet the Common Criteria Protection Profiles as for level two certification with the additional requirement of a trusted path, EAL 3 (or higher) and a Security Assurance Requirement of an informal Target of Evaluation security policy model [23]. As for level two certification, a trusted operating system may be used if a trusted path implementation protects plaintext critical security parameters.

Level 4 is the highest level available. To conform to this level the module has to monitor environmental conditions such as voltage and temperature as well. Fluctuations beyond a certain tolerance range has to trigger mechanisms to keep the device in its normal operating range or zeroization of plaintext critical security parameters (CSPs). Again, the software/firmware of the security module can be executed on a general purpose computing system if and only if the operating system is evaluated to Common Criterias EAL4 (or higher). And again, a trusted operating system may be used which is equivalently evaluated.

3.1.3. ISO/IEC 19790

The first ISO/IEC 19790 norm [34] was done in 2006 by the International Organization for Standardization (ISO) and the International Electrotechnical Commission (IEC). The standard labeled *security requirements for cryptographic modules* and is derived from the FIPS 140-2 standard released in 2001 (see Section 3.1.2). A small correction was published in 2008 [45].

No technical changes have been performed during transition. The reason for the transformation to an ISO standard was the missing ability to use non-US standard cryptographic algorithms as well [112].

3.1.4. Summary

While Common Criteria provides evaluation of a wide range of electronic devices, FIPS 140 and accordingly ISO/IEC 19790 are made for evaluation of security modules only. Whereas the US standard FIPS 140-2 is the most important standard for security modules in general. FIPS 140 itself points to Common Criteria for certain configurations. Therefore, the standards listed here are more or less equally important.

3.2. Fields of Application

The main goal of security modules is to hide certain key data from the public. Thus, security modules (SMs) can be used in a wide range of application. In the following, an incomplete list of example applications is given.

Communication The usage of public communication lines is not secure at all. Point to point encryption hides the transmitted data from a third party and therefore enables the secure use of public communication lines such as the Internet.

Encryption/decryption has to be done at both sides of the channel. Therefore, key material has to be exchanged (manually or automatically) and maybe saved for future use. This is where SMs can be used to keep sensitive key data protected and therefore provide a higher security level.

Storage "In today's computing environment, there are many threats to the confidentiality of information stored on end user devices, such as personal computers, consumer devices [...], and removable storage media" [105]. Again, the burden of security is shifted from the data to the keys.

Persistent storage of these keys gives the ability to decrypt the data later. This is where SMs can serve as secure key storage. Their use results in a much higher security level as if certain keys are simply stored in plaintext on some sort of media.

Digital Signatures Common routines like calling someone on the phone, watching television, reading the news as well as meetings and even signing documents are done more and more with help of computers. Therefore, cryptographic services, digital signatures in particular are getting more and more important. They provide data integrity as well as data authentication with the help of asymmetric cryptography.

Again, keys have to be persisted for long terms and of course kept private. And again, security modules are a neat solution to keep key data hidden. The Austrian Citizen Card concept [5] is a current example for the use of dedicated security modules for creating fully qualified signatures.

Public Key Infrastructure To provide a comprehensive and user-friendly use of digital signatures and encryption, PKIs are used. If a PKIs root key is compromised, all certificates issued by this PKI may be compromised. Further, every certificate issued by a PKI which was certified by the first PKI may be compromised. Therefore, protection of a PKIs root key is essential.

To shield the PKIs root key a security module serves well again. Its application complicates (unauthorized) access to the key and therefore boosts the overall security of the whole PKI to a higher level.

3.3. Cryptographic Services

In this section a listing of cryptographic services provided by modules of the market analysis is given. RSA, Data Signature Algorithm (DSA) and Diffie-Hellman Key Agreement Protocol

(DH) as well as their elliptic curve equivalents Elliptic Curve DSA (ECDSA) and Elliptic Curve Diffie-Hellman Protocol (ECDH) are supported as asymmetric cryptographic algorithms. Thales products also support El-Gamal and Korean Certificate-based DSA (KCDSA) [113].

Supported symmetric encryption schemes are AES as well as its predecessor Data Encryption Standard (DES) and Triple-DES (3DES). Further, RC2, RC4, and RC5, Koreas ARIA [67] and SEED [92], Canadas CAST [64] as well as International Data Encryption Algorithm (IDEA), Secure And Fast Encryption Routine (SAFER), Twofish, Blowfish, and Camellia are offered by selected modules.

For hashing, the standardized SHA-1 as well as the SHA-2 family are supported by almost all modules. Further, the less secure MD2 and MD5 algorithms as well as RIPEMD, HAS-160 and MDC-2 are offered by selected modules. For HMAC creation the available hashing and encryption algorithms are used.

All except two modules of the market analysis support pseudo and/or real random number generation. One of the standards ANSI X9.17 and ANSI X9.31 published by the American National Standards Institute (ANSI) as well as the FIPS 186-2 standard are supported by different modules.

3.4. Key Management

The reviewed modules (Table A.1) offer a great diversity of key management methods. In this section a summarization is given.

3.4.1. Generation

All modules except two of the market analysis ship with a certified RNG. Those are used to generate random data for all kinds of use such as nonces, one-time-pads or symmetric and asymmetric cryptographic keys.

3.4.2. Import/Export

17 modules support import of external key data, 11 devices support key export. There are different strategies in use.

All software SMs of the market analysis accept plaintext keys for import. Besides imported key, they also export generated keys in plaintext. Key wrapping (see Section 2.1.2) and storage is left to the application.

Other modules support wrapped export and therefore wrapped import of key data. The well protected module root key is used as root key for wrapping (see Section 2.1.2). Thus, a cryptographic key is generated, used and wrapped inside of the module. In other words, the plain key is never disclosed.

One module of the market analysis, the ARX PrivateServer [4], accepts plaintext key import if and only if the plaintext key is digitally signed by a certain authority.

3.4.3. Storage

There are several types of storage strategies for security modules. Mostly, software security modules do not offer long-term key storage. Advanced hardware SMs do. Long-term storage

keeps its data throughout power-off. In between there are other strategies.

There are modules which keep keys in volatile memory while in use. This method is used by plain SSMs, which are not capable of storing sensitive data at all as well as by high security SM which prevent storing keys on persistent media.

Other modules only store their root keys and user authentication data in a non-volatile way. Constants in program code which are loaded to RAM at module startup and buffered RAMs are used to achieve this functionality. The most robust SMs store their root keys on external hardware tokens such as SmartCards or Universal Serial Bus (USB)-devices. Therefore, the root key data has to be reloaded from the token on every startup and kept in RAM while power-on.

Furthermore, there are basically two ways of storing sensitive key and authentication data. One strategy is to store the sensitive data in plaintext. This makes sense if there is no way to access the modules memory as it is the case with an external and well protected dedicated hardware security module. Given that, the data can be used without any overhead and cryptographic services like encryption or signature creation can be performed faster. The alternative strategy uses cryptography to wrap sensitive data (see Section 2.1.2). Wrapped key or authentication data can be stored on insecure media such as public accessible hard drives, flash discs or optical storage. This method is used by modules which are plain software as well as resource constraint SMs like SmartCards or modules without any shielded non-volatile storage.

3.4.4. Zeroization

For keys stored only in volatile memory, zeroization is implicitly done by power-off. In reality, keys remain mostly readable. Further, keys are zeroed when they are not needed anymore. Modules with shielded persistent key storage zero their persistent keys on demand (command from operator) or on intrusion detection alert (if available).

3.5. Communication

Four different schemes are used by the reviewed modules. The boxes labeled “Application” represent the piece of software (an executable) which uses a certain cryptographic service. The operating system represents a common operating systems like Microsoft Windows, Apple Mac OS or a Linux distribution, as well as operating systems for mobile phones, PDA’s or other products which provide basic I/O operations for the application. The platform reflects a certain hardware platform such as a personal computer, a mobile phone, or some sort of micro controller. The red lines represent communication lines such as inter-process communication, software to hardware communication such as the Peripheral Component Interconnect (PCI) bus, or the USB as well as inter-network communication.

For SSMs there are two different schemes. The first solution uses libraries which provide structures and methods for cryptographic services (Figure 3.1a) within the application. Thus, the virtual (volatile) memory of the SM is the same as the virtual memory of the application. The second software solution is a dedicated piece of software (Figure 3.1b). The SM is an independent application accessible via a certain API. Thus, the SM is separated from other

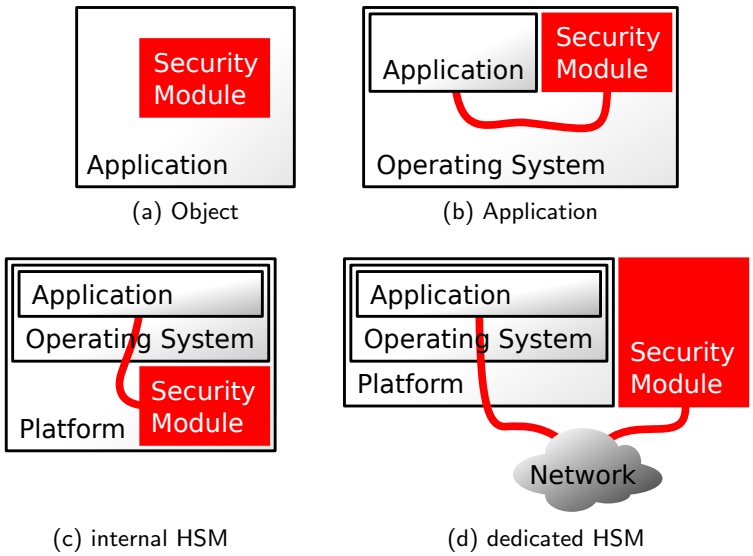


Figure 3.1.: Common architectures for use of security modules

applications by the operating system in respect to virtual address space (virtual memory) and execution (CPU).

A much more decoupled scheme is found with dedicated HSMs. They consist of a piece of hardware, the platform, and some software called firmware. The only way to communicate with those modules is a dedicated I/O channel. The Ethernet port I/O are used for external high security security modules (Figure 3.1d), internal devices are built into a common personal computer using the PCI bus (Figure 3.1c). Furthermore, devices connected via the USB and serial ports (Universial Asynchronous Receive/Transmit (UART)) and even protocols like T=1 [31] for SmartCard communication are in use. This scheme separates the security module from the calling application in a physical manner.

3.6. Authentication

The reason for using more than one authentication mode is to select a security level and to provide different roles with different privileges. Different types and role realizations spotted during our market analysis are discussed in the next sections.

3.6.1. Modes

Basically, there are three different authentication modes used by modules of the market analysis.

The first and simplest mode in use is the no-authentication mode. One can use certain services provided by the security module anonymously.

The second mode uses authentication based on knowledge of a secret value. There is one user role which is authenticated with a PIN. Knowledge of the secret PIN gives the opportunity to use the cryptographic services.

The third mode uses identity-based operator authentication. Each user has its own login name and password which is required to successfully authenticate to the security module. Alternatively, a few modules allow authentication using login name and a valid digital signature on a challenge placed by the security module. Furthermore, dedicated hardware devices are used for authentication on a few high security security modules. Authentication is just possible if one has access to a certain hardware dongle such as an USB-token or a SmartCard.

3.6.2. Roles

There are basically three different roles realized in SMs. The first role and also the simplest role is the anonymous user role. Almost every SM has this role. Many SSM use this mode for all offered cryptographic services. This is a legitimate way because no sensitive data is stored inside those SMs and therefore no harm can be done. High security HSM however use this mode for services like self-testing and status retrieval.

Another important role is the authenticated user role. This role is meant to access the modules cryptographic services as well as the users key store (if available on the SM).

The administrative role (crypto officer according to FIPS 140 (see Section 3.1.2)) is designated to manage the SM. The privileges contain initialization of the module, user management and backup as well as log viewing and zeroization of sensitive data.

A few modules of the market analysis support a multi-admin role to perform critical operations such as data migration or restoring of key data as well as booting the SM.

3.7. Backup and Disaster Recovery

Although backup and recovery is a rather important thing there is not much documentation about used techniques.

The reviewed software security module do not have a backup option. Key storage is done by the calling application. Thus, key management including zeroization and backup/restore has to be done by the calling application.

Modules without a build-in persistent key storage export generated keys only in wrapped form. So backing the wrapped keys up as well as restoring them is left to the host system. The master key of those modules can be backed up using hardware devices such as USB-tokens or SmartCards. Techniques like splitting keys among multiple hardware tokens are used to gain security. Restoring a key which is distributed among multiple tokens is possible if there is a certain number of tokens available.

HSMs with integrated key storage often support encrypted import/export of their key database, whereas backing up their root key is also done with hardware tokens. Furthermore, hardware-to-hardware device cloning, given a second identical device, is supported by some HSMs.

3.8. Summary

In this chapter an overview of services provided and techniques used in off-the-shelf security modules was given. First, the most important standards being the Common Criteria, the FIPS 140, as well as its successor ISO/IEC 19790 were discussed. Second, fields of application

were covered. Cryptographic algorithms provided by modules of the market analysis as well as key management procedures are stated as well. Communication between user and security module and authentication of users to the security module followed by a brief discussion of backup and disaster recovery strategies close the chapter.

The information gained in this chapter give an idea on how a security module is used, how it works in principle, and which challenges are to be resolved. Thus, basic requirements as well as inspirations can be derived for our TvSM design.

4. Design

Given the information collected in the previous chapters, this chapter reflects the TvSM design itself. We discuss selected challenges of module design, available solutions and the solution used in our presented TvSM. We do not cover aspects with straight-forward solutions in this chapter. For more information on these see the implementation part of the thesis (Chapter 7).

First, we discuss the most important functionality for a security module, the key management. We also cover different aspects of key management, be it necessary key types, master key management or key protection. Next, we cover other services which are not directly connected to key management. That is key migration, key backup or the time stamping ability for example. Then, we discuss the basic architecture of the presented security module. Last, we deal with operational techniques used for tasks like user authentication, module identification and communication.

4.1. Key Management

The main field of application for a security module is key management. A secure and easy key handling is crucial for a security module. During the TvSM design, we looked at different aspects of key management. Besides the standard requirements such as key creation and usage we evaluated other more special challenges.

In the following, we describe the inner workings of the key hierarchy. Further, we identify key types necessary for TvSM operation. Then we cover the special treatment of the master key as well as the use of key slots as well. Last, we discuss the need for key blob protection and possible realizations.

4.1.1. The Key Blob

In the TvSM context, a key blob contains the following components: The main component is the key data, obviously. The key data contains public key data, private key data as well as algorithm specific key parameters. Further, key attributes are included. They hold information about the type of the key and usage constraints. We give more detail on this in Section 4.1.4. The key secret protects the key against unauthorized use. If and only if the key's secret is provided correctly, the key can be used in cryptographic operations. Last, a checksum is part of the key blob. The checksum allows an integrity check of the key before use in cryptographic operations. We cover this in more detail in the next section.

4.1.2. Key blob Protection

A key managed by the TvSM is a blob of data containing key data, the key secret, the key type as well as other information. An attacker could attempt to simply pick a key and change

its data. For example, he can invalidate a key by changing a key pair's private part or he can change the key secret and therefore gain control over the key.

To prevent abuse, some sort of key protection has to be added. The first approach evaluated was to enable integrity checks by adding a keyed checksum to the key blob. The second approach was to enable integrity checks by adding a simple checksum to the key blob.

Keyed checksum

Adding a HMAC-based checksum calculated on a concatenation of the primitives to the key blob allows an integrity check prior to key loading. The key supplied to the checksum can be the private part of the parent key as well as the key's own secret. The final key blob looks like this:

$$keyblob = (K^{pub}, E_{K_{parent}}(K^{priv}), E_{K_{parent}}(KS), t, \ldots, HMAC_S(K^{pub}||K^{priv}||KS||t||\ldots)$$

with K^{pub} and K^{priv} being the key pair K, KS and t the key secret and the key type, $E_{K_{parent}}(\cdot)$ the key wrapping using the parent key K_{parent}, $\cdot||\cdot$ representing bit-wise concatenation, and S being the key used for the HMAC calculation.

The advantage of this approach is that the integrity check can be performed outside of the TvSM, assuming knowledge of the HMAC secret. The disadvantage is the required HMAC secret which has to be managed somehow. Reusing an existing secret, the key secret or the parent key secret for example, forces extraction (decryption) of the secret prior to the integrity check. In case of a secret which is not directly associated with the key, the secret has to be extracted from another source. Either another key has to be loaded or the secret has be provided in another way. Further, error correction is not feasible within this approach.

This approach meets the requirements but needs another secret whose security impact is hard to estimate. Thus, we did not used this solution for the presented TvSM design.

Checksum

Adding a simple cryptographic hash-based checksum to the key blob which is calculated on a concatenation of the keys primitives also allows an integrity check. On key creation, the checksum is calculated and added to the key blob. When the key is to be swapped out the key is wrapped using its parent key. The checksum is left world readable. The process of key wrapping blinds all critical data. That makes it impossible to recreate the checksum until the key is unwrapped (loaded) again. The keys blobs integrity can now be verified by recalculation of the checksum and comparison to the world readable one. The final key blob looks like this:

$$keyblob = (K^{pub}, E_{K_{parent}}(K^{priv}), E_{K_{parent}}(KS), t, \ldots, H(K^{pub}||K^{priv}||KS||t||\ldots)$$

with K^{pub} and K^{priv} being the key pair K, KS and t the key secret and the key type, $E_{K_{parent}}(\cdot)$ the key wrapping using the parent key K_{parent}, $H(\cdot)$ representing a cryptographic hash function, $\cdot||\cdot$ representing bit-wise concatenation, and S being the key used for the HMAC calculation.

The advantage of this approach is that there is no additional data needed to ensure data integrity. The disadvantage is that the integrity can be checked after unwrapping is completed at the earliest. Further, error correction is not possible with this solution either. Due to the simplicity of this approach we used it for the TvSM design.

Summary

In this section we stated two approaches which we reviewed during TvSM design. First, we discussed the keyed HMAC approach which needs an additional secret. Second, we explain the hash-only approach. The HMAC approach adds a keyed HMAC value calculated over the critical values of the key blob to the key blob. The hash-only approach adds a cryptographic hash value calculated over the critical values of the key blob to the key blob. Clearly, the latter approach is the simpler one and we therefore used it in the TvSM design. Both approaches allow key blob verification before key usage.

4.1.3. Key Hierarchy

A hierarchical key structure arranges the keys in a tree-like structure. Therefore, a rather versatile handling of key data is possible where key wrapping (see Section 2.1.2) protects the keys from disclosure. For example, the TPM as well as other security modules use the hierarchical key structure approach. For our TvSM design we evaluated two approaches. One is to support key which are maintained by the TPM (subsequently referred to as TPM keys) and keys which are maintained by software (referred to as software key) in the key hierarchy, the other is to support just software keys.

TPM Keys and Software Keys in Key Hierarchy

Supporting TPM keys in the TvSM's key hierarchy promises high security keys for high security applications.

An arbitrary key can wrap either a software or a TPM key. Cryptographic operations are done by the TvSM or the TPMs hardware respectively. An illustration of a mixed key hierarchy is given in Figure 4.1.

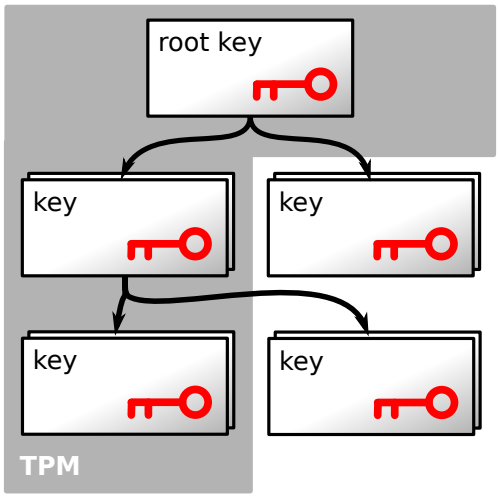


Figure 4.1.: Mixed key hierarchy approach

This approach allows creation and usage of hardware protected keys in a hardware protected environment, namely the TPM.

In other words, by using a TPM key, the TvSM serves as an interface to the underlying TPM. From the operators point of view the TvSM turns into an HSM for that special key. The downside of using a TPM key is the weak performance (see Section 6.2) as well as the (very) limited set of algorithms. Further, this approach has disadvantages in key handling. For example, the weakest key in the hierarchy above a key determines its security level. Thus, the security gain holds if and only if the parent key of a TPM key is a TPM key as well. Given that, a chain of TPM keys has to be maintained which is not at all user-friendly. Further, a TPM key cannot be designated for key wrapping and bulk encryption at the same time. Designation to key wrapping and data sealing is possible. Therefore, one solution is to maintain separate key trees. Another solution is to use data sealing. The downside of this approach is that whenever the TvSM software is altered (updated), the system state has changed and therefore, the sealed keys are no more readable. So every time the software changes, every key has to be rewrapped and therefore resealed. This complicates key handling even more.

All in all, the disadvantages regarding usability, speed, and the lack of algorithms to choose from outweigh the advantage of higher security. Therefore, we did not use the approach of a mixed key hierarchy in the TvSM design.

Software Keys in Key Hierarchy

Supporting only software keys in the TvSM's key hierarchy promises simple usage but lower security compared to hardware key shielding solutions.

An arbitrary software key can wrap another arbitrary software key. Cryptographic operations are done in software. The hierarchical structure therefore contains only software keys.

The advantage of this approach is its simplicity. There are neither structural nor computational limitations such as constraints for parent keys or algorithm type limitations respectively. The disadvantage is that keys cannot raise above a certain security level. The TvSM shielding mechanisms cannot reach the same level of protection as those of a dedicated HSM.

Allowing just software keys in the TvSM key hierarchy is a simple solutions which allows TvSM operation. The disadvantage of a less secure key shielding ability compared to hardware solutions is outweighed by the advantages. Thus, we used this approach to implement the TvSM's basic functionality.

Summary

In our TvSM design we only use operations based on software keys except for root key handling which is discussed later. Advantages are support of a broad range of algorithms as well as easy hierarchy maintenance. TPM keys would require special treatment and algorithm variety is rather low, in fact its just RSA today. The much more complex maintenance and usage is not worth the advantage of a better key protection. If better key protection is needed, other products might suit the needs better.

4.1.4. TvSM Key Types

For a key's security it is crucial to limit its usage to a specific cryptographic service [122]. Either encryption or signature creation, for example. Given that, limitations of key usage have to be defined. For TvSM design that is a storage and a signing key type.

The storage key is limited to the encryption service. More specific, the storage key is intended to wrap other keys. No bulk encryption is allowed. The wrapped keys can be of any type in terms of usage limitation as well as in terms of algorithmic type. The signing key type is limited to the signature creation service. They are only used for signing operations, be it a signature on bulk data or a time stamp. They can neither bulk encrypt data nor wrap other keys. Other keys, keys for bulk encryption for example, are possible as well but not needed for the TvSM design. Nonetheless, we prepared the TvSM for such an enhancement.

To sum this up, the low number of different areas of usage leads towards a simple system on the one hand, on the other hand it limits the services available to just signature creation. Nonetheless, this meets the requirements well.

4.1.5. The Masterkey

Given the hierarchical key structure defined in Section 2.1.2 and Section 4.1.3, the root key is a key which needs special treatment. The root key is the most important key of the whole structure. Disclosure of the root key means disclosure of every other key in the hierarchy. In the context of security modules this root key is called master key and is protected and managed by the security module itself.

In the following, we discuss persistent storage solutions for offline key protection as well as management operations and their constraints for key protection while power-on.

Persistent Storage

It is crucial to prevent the root key from disclosure. Therefore, a number of solutions were evaluated during the TvSM design: storage on portable media, in the TPM's NVRAM and on the hosts hard disk.

Portable Media Portable media such as SmartCards and USB flash disks are used for storing master keys of high security HSMs. Often, a number of n portable devices out of m are needed for master key recreation. Such schemes are discussed by [68] and [107], among others.

The master key is created by the TvSM and stored on one or more flash disks. Whenever the module boots up, the key has to be transfered to the TvSM. This would require two things: first, a secure communication line between the flash disks and the TvSM; second, some authentication data provided by one or more administrators which forecloses unauthorized key transfer. After transfer, the TvSM holds the key in Random Access Memory (RAM) until power-down. An illustration of the resulting key hierarchy is given in Figure 4.2a.

This approach is highly secure. One must get hold of one or more flash disks plus one or more authentication secrets to disclose the master key. This is hard if the flash disks are kept separately and in safe places. Further, solutions based on the Byzantine Generals Problem [91] exist to handle one ore more rogue administrators without key disclosure. The disadvantage is the operational complexity of the approach. Whenever the module is to be restarted a complex procedure has to be accomplished to recreate the master key without being tampered.

At large, this approach is suitable for highly secure security module in very critical environments. It is not suitable for a low-cost and easy to use application like the TvSM. Thus, we did not use portable media as storage location for the module's master key while power-off.

TPM NVRAM The TPM has a shielded non-volatile RAM which can be used by applications. That promises a highly secure location for persistent master key storage.

The TvSM generates a master key and stores it in the TPM's NVRAM. Every time the TvSM is fired up, it reads the master key from the TPM and holds it in RAM until power-off. An illustration of the resulting key hierarchy is given in Figure 4.2a.

A great advantage is that given this approach the master key never leaves the TvSM. In power-off mode it is protected by the TPM. That makes the approach a highly secure solution. An advantage over the approach using portable media discussed before is that no

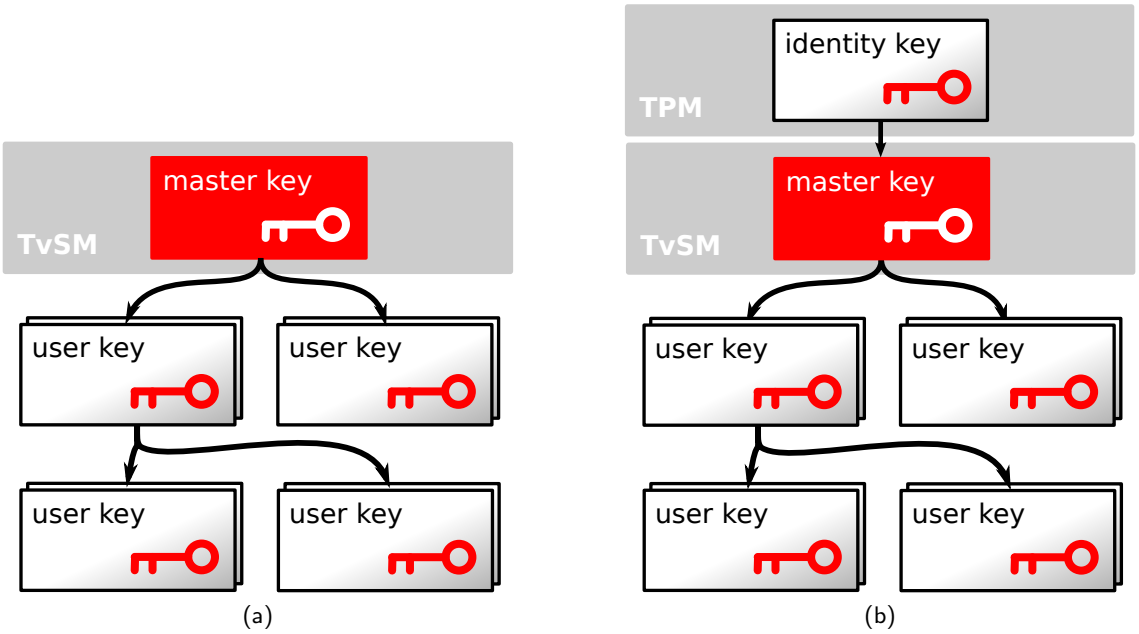


Figure 4.2.: Evaluated key structures

complex procedure has to be accomplished for module startup. The downside of the approach is the very limited size of the TPM's NVRAM (a few kilobytes). The chance of conflicting other applications is rather high.

At large, the solution is arguable. The limited size of the NVRAM which directly limits key size scalability of the TvSM made us not to use this approach in TvSM design.

On Disk Protecting a root key through encryption is not possible. This step would move the root key down the hierarchy and the key used for encrypting the former root key would be the new root key. Introducing another key for master key protection is only reasonable if the new root key can be protected with less effort and if the key can be used for other purposes as well.

It could work as follows. The master key is wrapped by another key. The wrapping key is a TPM key limited to be non-migratable and for binding use only. With that, the master key can be wrapped, swapped out, and stored on the hosts publicly readable hard drive. The TPM key used for wrapping the master key is later referred to as the module's Identity Key (ID key) (see Section 4.4.3 for more details). When the TvSM is started up it unwraps the master

key stored on the hosts hard drive and holds it in RAM until power-off. An illustration of the resulting key hierarchy is given in Figure 4.2b.

This approach has the following advantages. Being wrapped by the ID key, the master key is under the protection of a HSM, namely the TPM. Further, the ID key and therefore the master key is bound to one specific TPM. It is therefore impossible to copy the master key to another (compromised) TvSM, bypassing the secured migration process. The ID key cannot be moved due to its usage constraints. Another advantage is that no limitations regarding key length or algorithm type is given. In other words, the key could be of any type and any length which guarantees scalability for future designs. Beyond that, for the user the master key still appears as the root key of the hierarchy. Even for TvSM startup no administrative interaction is necessary. A disadvantage arises in the fact that root key protection is shifted to another key, namely the module's ID key. This key requires special treatment as well.

The new ID key can be used for solving the module identification challenge which is discussed in Section 4.4.3 and therefore its existence and the need for managing another key is arguable. Thus, we used this approach for our TvSM design.

Management Operations

Given the master key stored within the TvSM demands management capabilities: basic tasks like creating and deleting the master key as well as more advanced tasks like migration and backup of the master key material.

For the TvSM design, master key management works as follows. Prior to any operation, a successful authentication for the privileged role (see Section 4.4.1) is necessary. This is taken as assurance that the person operating the master key management is allowed and qualified to do so. Further, the operations require different preconditions to be satisfied. To create a master key for example, the TvSM demands for no master key being set. For a successful master key deletion as well as for migration the master key's secret has to be provided. The master key's secret is set on creation. Master key migration allows backup of the master key on another TvSM. How migration works is discussed in Section 4.2.2.

These operations prevent disclosure of any keys of the hierarchy even if the master key's secret is disclosed. One can delete the key which leads to denial of service (DoS) or one can migrate the master key to another TvSM which shields the key as well. The only way of disclosure is migrating to a compromised TvSM.

Summary

In this section, we discuss persistent storage solutions as well as management operations necessary for the module's master key. During TvSM design we evaluated three different approaches for a non-volatile storage of the master key: portable media, the non-volatile RAM provided by the TPM, and a solution for hard disk storage. In our presented design we used the latter. We did that by introducing another key which is protected by the TPM, the module's Identity Key (ID key). The ID key wraps the master key and therefore protects the master key from disclosure. The ID key is used for module identification. We discuss module identification techniques in Section 4.4.3.

Master key management is needed to perform tasks like master key creation, deletion, migration, as well as backup. The authentication challenge as well as other constraints necessary

for administrative tasks were covered in this section.

4.1.6. Key slots

Keys are stored outside the TvSM, they are encrypted (wrapped) and therefore depend on their parents. To use them they have to be unwrapped prior to a cryptographic operation. Besides, they have to be shielded against disclosure. This task is done using key slots for temporary key storage. A similar approach is used for key handling within the TPM.

For each standard role session a configurable number of key slots are provided. The user asks the TvSM to load a given key. The module takes the key, unwraps it with the given parent and moves it into an empty key slot. A handle, a random string value which is ideally unique within the session, is generated and returned to the user. Now the user can use the key for cryptographic operations by specifying the key's handle. Unloading the key removes the key from the key slot and the handle is invalidated. On the end of the session the key slots and therefore the keys loaded are destroyed.

One advantage of key slots per session is the low chance of a handle collision and a subsequent misuse. On the one hand the module is kept free from unused key data, on the other hand prevents misuse of keys through disclosed or guessed handles by invalidating the handles on session end. A disadvantage is due to the limited number of available key slots per session, the user of the TvSM (an application) is responsible for key slot management.

At large, key slots are a neat solution which are used inside the TPM and other security modules as well. Therefore, we used this technique in TvSM design.

4.1.7. Summary

In this section the basic services offered by our presented TvSM are explained. First, it is discussed whether or not to include TPM keys in the key hierarchy. Including TPM keys in the hierarchy would lead to a higher security level. No TPM keys in the hierarchy result in a much easier key management and usage. Second, necessary key types and the need for a special treatment of the module's master key is covered. Three master key storage solutions are explained as well as management operations and their corresponding constraints. The idea and realization of key slots followed by a discussion about the necessity of key blob protection and the solutions conclude the section.

4.2. Services

Besides the key management capabilities the TvSM is intended to offer other services as well. We added the ability of key import and export, a key backup and migration solution, as well as a time stamping service. Due to their straightforward implementation, the signature creation service, the statistics collection and retrieval service, the logging service as well as the session management are discussed in the implementation part of the thesis (see Chapter 7).

4.2.1. Key Import/Export

The key import capabilities are meant for migrating existing keys to the TvSM. These keys may be managed by another security module or just by unshielded software. The export capability

works the other way round. Exporting a key allows disclosure of the private part of a key for use with other applications.

Given an existing key which is to be imported, the first step is to check for its compatibility with algorithms available within the TvSM, key lengths, and so on. On success the key is imported. Export works the other way round. Given a loaded key referred to by its key handle one can export the key unencrypted. Knowledge of the key's secret is necessary. The solution discussed up to now allows one to export a storage key and reimport it as a signature key. This property violates an important security requirement (see Section 4.1.4 for further details on key usage). To foreclose that, we introduced a number of constraints. To prevent key disclosure through a disclosed wrapping key, neither import nor export of storage keys is allowed. Further, only imported keys can be exported. That prevents disclosure of a key created by the TvSM. This constraint is the most conservative constraint available for key export because it does not allow any disclosure, be it migration to any other security module or migration to an unshielded application.

To sum this up, securing key import and export through constraints is a challenge where the solution is a compromise between comfort and security. We decided to limit the import and export service in the way stated above.

4.2.2. Key Migration

Moving keys from one parent to another parent is called key migration. It is a common practice for hierarchy maintenance and therefore used in the most security modules, including the TPM.

This is how it works. Key migration can be controlled for each key individually and is fixed at key creation. Given a migratable TvSM key to be moved to another parent, one has to load the key, memorize the key handle, then provide the key handle, a migration secret and the target key to re-wrap the target key with the target (new parent). The TvSM can be configured to use the key secret, the authorization secret used for authenticating the privileged role, or a concatenation of both as migration secret. The TvSM key is now rewrapped using the target key and swapped out of the TvSM. Using another arbitrary storage key as new parent key is a very simple and secure solution. Further, there is no difference between local and inter-hierarchy migration. Given the rewrapped key, it can be loaded under the new parent without further steps to be taken.

The migration secret is a secret which is already in use in our TvSM design. The advantage of this solutions is that neither an extra key, an extra secret, nor a complex process is needed to perform a key migration.

At large, the solution meets the requirements well. It solves the migration challenge as well as the master key backup challenge. Therefore, we used it for our TvSM design.

4.2.3. Master Key Backup

The scenario of a broken master key which affects all keys in the hierarchy is the worst case imaginable. Thus, key backup is an important part when it comes to key hierarchies. For example, Souza at al. has presented complex backup procedures for security module used in PKI applications [78]. While evaluation of different approaches the boundary between key backup and key migration blurred. Thus, we evaluated a backup-via-migration solution.

Given a second TvSM setup, the master key is migrated to the second TvSM. The process is illustrated in Figure 4.3. This requires authorization to the privileged role as well as the

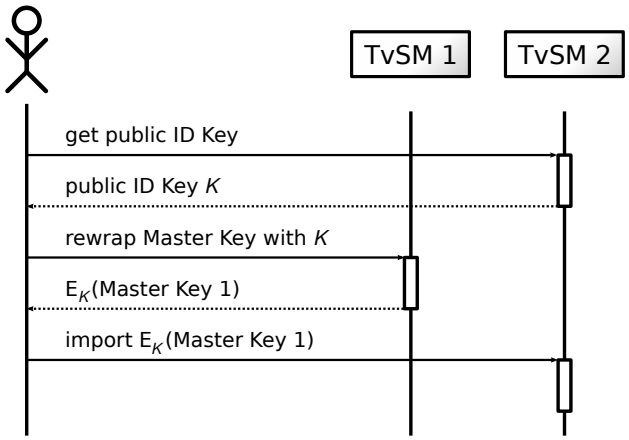


Figure 4.3.: Master key backup

master key secret as for all master key operations (see Section 4.1.5). The master key is now available on both TvSMs and therefore, a certain level of redundancy is achieved.

The advantage is the simplicity of the stated solution. No additional procedure has to be designed and secured which results in a smaller attack surface and simpler handling. The downside is the necessity for a second TvSM setup.

All in all, migrating the master key to another TvSM is a neat and secure solution for the master key backup challenge. The advantages clearly outweigh the disadvantages. Thus, we used the backup-via-migration solution in TvSM design.

4.2.4. Time stamping

The TvSM should also offer a time stamping service. Given a blob of data and a key, the TvSM should return a signed package containing the provided data blob and a date/time statement. Preferable, the TvSM should not use any Internet connection to keep the attack surface small. In order to provide that service two major of steps have to be taken. First, the TvSM has to be turned into a time stamping authority. Second, the TvSM has to be kept a time stamping authority once it became one. These two steps are now discussed in detail.

Time Authority

The first challenge is to transform the TvSM into a time authority. A time authority is a trusted third party which serves accurate time information. During TvSM design we evaluated different approaches, namely usage of an external time authority as well as utilization of a PC's build-in system clock.

Whenever a time stamp is needed a remote time authority is used. An online connection is obtained, a protocol like the Time Stamping Protocol (TSP) [65] or the Precision Time Protocol (PTP) [44] guarantees small latency and the time stamp can be performed. That is a clean and reliable solution. Nonetheless, it conflicts with the requirement of no Internet connection. Consequently, we did not use this approach in our design.

Another solution is the use of the system's build-in real time clock. This clock cannot be trusted as is. Threats like someone or something changing the clock, be it on purpose or not, imprecision due to power supply issues like a low battery, or even imprecision due to parameter fluctuation while factoring the clock can state inaccurate time information. Thus, we decided to introduce an approved mode for the system clock. Within the approved mode, the system's clock is a trusted source of time information. The approved mode is reached through administrative interaction with the TvSM. In other words, the administrator can ask the TvSM for a time statement. After verification of this time statement, the administrator can either adjust the clock or simply state that the clock is set correctly. After that, the TvSM accepts the system clock as a trusted source of time information and enables the time stamping service.

This solution requires a trusted administrator. A rogue administrator could approve any time and the TvSM has no chance to detect that. This might be considered as a disadvantage. A clear advantage is that this approach supports radio controlled clocks attached through the serial port or USB, to name a few.

Altogether, this is a neat and easy solution meeting the requirements. The major drawback is that one has to fully rely on the administrator. Nevertheless, this solution was used in the TvSM design.

Securing the System's Time

Given the system clock being in approved mode does not protect from the aforementioned threats like clock poisoning on purpose or not, due to parameter fluctuations during factoring or any other cause leading to an inaccurate time statement. Consequently, the approved system time has to be secured somehow.

For our presented TvSM design the following approach was used. The TPM offers a tick counter service which can be set up to conform International System of Units (SI) time spans. With that, a second independent source of time information is available. Whenever the TvSM sets the system's clock to approved mode, the value of the TPM tick counter is memorized. A check operation is performed periodically with a configurable frequency f. This operations compares the amount of time passed since the last check on either clock, the system clock and the TPM tick count. This is the time passed referred to the system's clock T_{SC} and the time passed referred to the TPM tick counter T_{TC}. When the discrepancy exceeds a (configurable) threshold σ, the approved mode is aborted and consequently, the time stamping service is disabled. The check-frequency f is set to $f = \frac{1}{T_{SC}}$ with T_{SC} being configurable. The mode for the next period is set to $approved$ if and only if

$$T_{SC} \in [T_{TC} \cdot (1 - \sigma), T_{TC} \cdot (1 + \sigma)]$$

.

The advantage of this approach is that any discrepancy between two independent clocks can be detected, be it caused by hibernation or a malicious clock reset. The disadvantage is that the discrepancy cannot be monitored live. Depending on the check frequency f a time frame $T_{SC} = f^{-1}$ is left open for attacks.

At large, this approach meets the requirements quite well. Given the drawbacks the time stamping service is not suitable for high security applications.

4.2.5. Summary

In this section services of the TvSM are discussed and described which are not essential for basic TvSM operation. Nonetheless, these services allow a greater field of application for the TvSM. First, the service of key import and export is covered. The service allows usage of keys which were not generated by the TvSM on the one hand, on the other hand the service allows keys which are generated by the TvSM to be used in other applications. Second, the challenge of key migration is covered followed by the challenge of backing up the master key. Backing up the master key is done through key migration. Key migration is done by a simple re-wrap of the key with a new parent. Finally, the time stamping service is covered. The challenges of how to create a time authority and how to secure the created time authority are explained in detail. The other services offered by our presented TvSM are discussed in the implementation part of the thesis (see Chapter 7) due to their straight-forward implementation.

4.3. Basic Architecture

There is no chance of tampering with an ideal security module. However, ideal security cannot be done in practice. Threats like hardware tampering or code injection/execution are to be withstood as well as API-level attacks [69][70][77][121], for example. The TvSM design has to satisfy requirements like code manipulation detection as well as resource isolation. We discuss the basic architecture we used for TvSM hardening in the following.

An illustration of the whole architecture is given in Figure 4.4. Being a piece of software, the TvSM server and the client need a host. For the TvSM service, the underlying platform has to meet two basic requirements. First, Intel TXT (see Section 2.3.5) has to be available

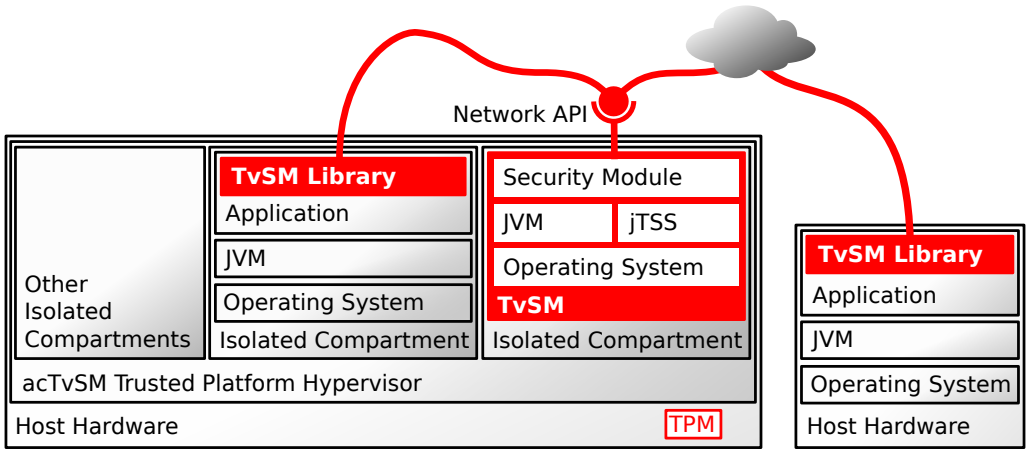


Figure 4.4.: Basic TvSM architecture

and with that a TPM, which is the second requirement. A client platform can be anything capable of a network connection and Java.

For TvSM server hardening we used the acTvSM platform [100][101][115]. The platform offers a variety of services. First, resource isolation through operating system virtualization is provided (see Section 2.2 for further details on virtualization). In other words, one cannot read or alter resources from another compartment. That includes volatile as well as non-volatile

memory. Second, the acTvSM platform offers encryption of data. In particular, operating systems images which are meant to be booted in isolated compartments are encrypted while powered-off. In other words, offline tampering is made hard. In the TvSM use case that means that critical data such as the ID key cannot be read externally, neither while operation nor while offline. That meets the resource isolation requirement quite well. Another service provided by the acTvSM platform is secure boot. A chain of trust is build using the late launch process available on an Intel TXT capable platform while booting an operating system image. Given a known-good state statement, an entity that is part of the chain of trust decides whether the booted operating system is trustworthy or not. Therefore, the booted operating system ensures a known-good software state right before the software is usable. Furthermore, the platform provides update management mechanisms which allow comfortable reconfiguration operations on compartment operating systems. Software update and bug-fix handling therefore become rather easy. The last service used by the TvSM is the acTvSM platform's ability for TPM forwarding. It allows compartment operating systems to use the hardware TPM without further management. Other services are offered by the acTvSM platform as well, but are not used in our presented TvSM. The interested reader is referred to [115] for further information.

For the TvSM's base operating system we selected a Debian Linux distribution [20] with network and Java enabled. We use two separate hard disk partitions for our presented TvSM. The first partition is marked read-only and holds the operating system and the TvSM code base. The second partition offers read-write-access and is used for storing logs and statistics as well as the module's Identity Key and the master key. The read-only partition is measured during the late launch process and therefore, the module's trustworthiness is decidable.

The TvSM itself is a plain Java application, reachable via a lightweight network-based API only. The TvSM library hides the inner workings of communication and authentication and is therefore needed to accomplish communication and therefore operational use of the services discussed in the sections before. More details on the Java implementation is stated in the implementation chapter of the thesis (see Chapter 7).

The client application has to use the TvSM library and therefore has to support network access as well as Java. Besides that, no limitations are given. The client application can be operated on the acTvSM platform inside an isolated operating system as well as on a remote host. That leads to simple configuration and usage of an application which makes use of the TvSM services.

To sum this up, the basic architecture offers a high level of isolation and protection to the data and software which form the TvSM. Further, it is easy to connect to the server given any client capable of a network interface. More on how connections are established and secured is discussed in the next section.

4.4. Authentication and Operation

Providing a basic set of key management tools as well as a set of services is necessary but not sufficient for module operation. In this section we discuss the missing building blocks for a working security module.

First, we identify roles followed by a discussion on authentication techniques. Further, we cover the challenge of identifying the TvSM as well as the communication channel between user and TvSM.

4.4.1. Roles

For the TvSM design, we defined two roles; a privileged role and a standard role. The privileged role corresponds to the crypto officer role and the standard role fits the user role which were discussed in Section 3.6. Further roles such as an anonymous role or a supervisor role were not defined.

The privileged role is designated to perform module related tasks such as master key management, statistics and log retrieval as well as time authority control. Therefore, the role is critical to security and trustworthiness of the TvSM. A security breach would affect the whole TvSM except user key data. Nonetheless, the privileged role has to be well protected against tampering.

The standard role is designated to perform user related tasks such as user key management. Knowing the corresponding key secrets, a user can manage and use existing keys. Further, the key creation task forces the user to supply a usage secret. The standard role does not have access to the module configuration. Given that, access to the standard role, obtained by regular login or by security breach, allows the use of the services offered by the TvSM but nothing more. In other words, neither access to the module's configuration nor usage of other user keys is possible. Given that, no authentication is needed for the standard role. Nonetheless, standard role authentication can be used to allow or forbid TvSM usage for a certain user.

4.4.2. Authentication Types

Given the role requirements discussed in the previous section, a suitable authentication method has to be found. Moreover, other requirements have to be met. First, a small attack surface is better than a large one. That requires a limited interface to the user providing only absolute essential operations. Second, usability should be high. Last, it is necessary to distinguish between the standard and the privileged role. Further, the standard role needs a less secure authentication technique than the privileged one.

In this section, anonymous authentication is discussed as well as role-based, identity-based, multi-identity-based and multi-factor authentication. We selected role-based authentication for our TvSM design.

Anonymous authentication

Anonymous authentication means that there is no user authentication at all. This technique is used for coffee dispensers as well as for Storage Root Key usage in the trusted computing context.

For realization, no login process and therefore no login data is necessary at all. Another realization is authorization using a world known secret. Anyone can use the all services provided.

The main advantage of this approach is its usability. No authentication data has to be distributed, no logins managed. A user can just use a provided service without the need for authentication data. The main disadvantage is that there is no limitation at all. There is no way of distinguishing between a standard role and a privileged role. Further, there is no way of distinguishing between one user and another.

All in all, anonymous authentication fits the standard role authentication requirements quite well. Further, it meets the requirements of ease of use and the small interface. However, it is not suitable for distinguishing between multiple roles. Consequently, it is not suitable as single solution for TvSM design. To meet the privileged role authentication requirements another solution would be necessary, which obviously causes a larger attack surface as well as a decreased usability. Therefore, we did not use this approach for the TvSM design.

Role-based authentication

Role-based authentication is a simple solution used in various applications. It is used whenever it is important to distinguish between different privilege levels and not between different individuals.

For realization, two different logins are necessary, one leading to the privileged role and one leading to the standard role. The login data leading to the privileged role has to be shared among all administrators, the login-data leading to the standard role among all users. The TvSM role mapping, authentication and authentication data is illustrated in Table 4.1.

Role	**Authentication**	**Authentication data**
standard	operating session	TvSM master key, operating secret
privileged	maintenance session	TvSM Identity Key (ID key), ID key secret

Table 4.1.: TvSM role mapping and authentication data

An advantage of this solution is the simple login management. Just two different logins are needed. Therefore, it is possible to distinguish between two logins leading to two different privilege levels. The disadvantage is that login-data has to be shared among a certain number of users. That leads to a certain level of anonymity within a group of users. For example, in case of a security breach caused within the privileged role, it is not possible to pin down the responsible user given a group of users which have access to the privileged role. Given just one administrator, he can configure the module and distribute the login-data at his own will. There is no way of supervision and control.

At large, role-based authentication fits the privileged role authentication requirements as well as the alternative requirements for standard role authentication well. Further, its an arguable trade-off between usability, simplicity of the interface and the security level reached for role authentication. More complex authentication techniques can, if required, be managed by the application using the TvSM. Therefore, we used role-based authentication for TvSM design.

Identity-based authentication

Identity-based authentication allows different identities to log into a system. This scheme is used with operating systems as well as with cash dispensers. In both cases one has to provide a login name, be it a sequence of characters or a SmartCard and a corresponding password, another sequence of characters or a PIN, for example.

In the TvSM context, identity-based authentication would work as follows. Each and every user has its very own account with his own authentication data, a login name and a secret.

Each account is assigned to one role, standard or privileged. In other words, the authenticated user either operates the standard role or the privileged role.

The advantage of this solution is that every action is associated with a user. Therefore, given the scenario of a rogue administrator, the action which causes the security breach can be traced to the responsible user. Another advantage is that no secrets have to be shared among a group of users. That allows a strict access control whether a user is allowed to use the TvSM's services or not. The downside of an identity-based authentication scheme is that the logins have to be managed by the TvSM. Operations like login creation, deletion, password resets and so on have to be provided. For simple usage user accounts have to be manageable online and without service interception. Given that, privilege escalation attacks (a user can grab more privileges than he is limited to) for example could arise against the TvSM.

Altogether, identity-based authentication is a suitable solution as well. It allows a more precise user management which also pays off in user supervision capabilities. The downside is the greater amount of logins which result in a more complex login-management. The interface has to be enhanced to provide those configurations and management operations. That clearly conflicts the requirements of a lightweight user interface and therefore usability and a smaller attack surface. If identity-based authentication is necessary for a special application the functionality can be moved to the application itself. Consequently, we did not use the identity-based approach for the TvSM design.

Multi-identity-based Authentication

Multi-identity-based authentication is a technique used for high security applications such as high security hardware security modules. It requires more than one user for login to a specific role. In general, an n out of m authentication is required, where n states the required number of successful authentications and m the total amount of available authentications. More about multi-identity authentication can be found in [75] and [82].

To gain access to the privileged role, n out of m administrators have to provide their authentication data. This approach overshoots the requirements of standard role authentication.

The advantage of this solution is that at least n administrators have to join forces to use the privileged role. In other words, a certain level of control is added. Only a group of n or more rogue administrators can abuse the privileged role. The disadvantage is the necessity for a group of administrators with individual authentication data. This again requires a certain amount of user management capabilities and all the challenges discussed in the previous section. Another disadvantage is the complexity of the authentication system which goes beyond the technical layer. A complex management of authentication data as well as the authentication process itself is necessary.

All in all, the solution is suitable for the TvSM design for privileged role authentication. It offers a high protection level to the privileged role. Again, an interface extension is necessary to provide the service. This again conflicts with various requirements such as ease of use, a small interface and a small attack surface. If this functionality is necessary for a certain application it can be realized in the application using the TvSM. So, we did not use the multi-identity-based approach for TvSM design.

Multi-factor Authentication

Multi-factor authentication uses different authentication sources for user authentication. An example is a bank card used at a cash dispenser. You have to have the bank card and the PIN available to withdraw any cash. More information on multi-factor information can be found in [97].

Due to the TvSM's software nature, no secure channels in hardware are available. That means, no card or fingerprint readers, for example, can be attached to the TvSM directly. Remote usage of such devices is possible.

The obvious advantage is that the chance of an attacker being capable of all required authentication inputs is lower for more than one input as for just a single one. The downside is the more complex management of the technique.

Multi-factor authentication is suitable for the TvSM design as well. It results in a higher security level but also a more complex management and usage. Nonetheless, this approach conflicts with the requirements for usability, a lightweight interface and a small attack surface. Again, the functionality can be provided by the application which needs the functionality. We did not use the multi-factor authentication technique in our presented TvSM design.

Summary

In this section we discussed authentication types which we evaluated during the TvSM design process. Anonymous authentication does not allow role separation, the identity-based approach as well as the multi-identity and multi-factor approach cause too much overhead for our TvSM design, be it a complex usage or the need for additional hardware. Role-based authentication is the best solution for our TvSM design. A more complex technique can be provided by the calling application.

4.4.3. TvSM Identification

Using a compromised security module could be fatal for key security. Key secrets as well as keys could be disclosed and secrecy of private keys is therefore no longer given. A single physically connected module is rather easy to identify. It is more difficult when access is made available via a network. In the latter situation, impersonation, for example, becomes a rather simple task. Generally, identification is a very important and crucial task in module operation.

In the following, different techniques of module identification are listed and discussed. All of them where reviewed during the TvSM design. The final solution is discussed as well.

Pre-shared Secret

An intuitive way of identification are secrets pre-shared between the one who wants to identify a certain device and the device itself. This principle is successfully used in other services like PINs for credit card authentication or Transaction Authentication Numbers (TANs) for transfer authorization in electronic banking for example.

The technique adapted to the TvSM context could work as follows. Each TvSM holds a secret which is unique across all shipped TvSMs. Each shipped TvSM contains a printed version of its secret. The printed version can be scrambled or blinded to prevent eavesdropping by unauthorized eyes. Previous to module operation one can verify the module's identity by

retrieving the module's secret from the TvSM installation, followed by a manual comparison to the printed secret shipped with the TvSM. If the values match, the module is the correct module.

This technique has a couple of disadvantages. The unique secret has to be injected into the shipped disk. That means, every single software module has to be supplied with a pre-shared secret prior to disk finalization. Further, a printed version of the secret is needed. Each TvSM disk has to be enclosed with its corresponding printed secret. Secret injection as well as matching disk and printed secret will cause additional effort to production and packaging. Another disadvantage concerns usability. Comparison of the secrets and therefore identification of the module itself is a simple task. But prior to that, the printed secret has to be hidden at some safe place. For comparison, it has to be disclosed and finally the secret has to be retrieved from the module. Only now the check can be performed. That makes usage a rather complicated task. Another disadvantage is that an attacker could retrieve the secret from the module as well and inject it into a compromised module.

All in all, this technique is rather complex in terms of packaging and shipping, complex in terms of usability and it can be attacked very easily. Therefore, it is not suitable for module identification and was not used in TvSM design.

PKI/Certificates

Certificates and therefore public key authentication (see Section 2.1.2 for the basic idea of public key infrastructures) can be used for module identification also. This technique is used by the Transport Layer Security (TLS) protocol [79] for securing Internet communication, among others.

Each TvSM holds its very own certificate. That implies that a shipped TvSM instance has a private key and a corresponding world readable public key. Given that, the key pair can be used for module identification. A realization can be digital signatures, another session key creation. For signatures (see Section 2.1.2 for further details on digital signatures), prior to operation, the user sends a challenge (a nonce) to the security module. The module signs it with its private key and sends it back to the user. The user now can verify the signature. On success, the module is the correct module.

This technique has advantages over the previously discussed pre-shared secret solution but also disadvantages. Like for the pre-shared secrets technique some unique data, the private key and the corresponding certificate including the public key, has to be injected into each instance prior to disk finalization. But, there is no need for instance specific printed information like secrets or keys. Since there is no need for matching sheets and disks, packaging gets simpler. Another advantage is that the use of digital signatures for module identification does hardly allow tampering, assuming that the private key cannot be retrieved from disk. In other words, an attacker cannot retrieve information from the running module or the shipped disk to impersonate the module by some compromised installation. Another advantage is that usage gets simpler. The certificate can be distributed to everyone who wants to use the security module. It can be stored on insecure media such as the system's hard drive and therefore has not to be fetched from some secret storage location.

Altogether, this technique is more elegant in terms of usability, it is safer than the pre-shared secret alternative and simpler in terms of packaging. Nonetheless, keeping the injected private

key secret is a tough part. The key has to be available in plaintext for signature creation which implies that it has to be stored in plaintext format. Even if the key is encrypted, the encrypting key has to be stored unencrypted. By implication, one key has to stay unencrypted which allows retrieval of the private key and therefore this solution is not safe. Consequently, this solution is not suitable for module identification neither.

Certificates and the Trusted Platform Module

Utilizing the TPM of a platform for module identification promises a higher security level. Here we discuss a possible combination of the certificate solution discussed before and the tamper resistant storage capabilities of a TPM.

The technique could be adapted to the TvSM context as follows. Given a certificate and its corresponding private key injected prior to packaging, a security module can hand its private key over to the protection of the TPM. Whenever a user makes an identity check request, there are basically two ways to create the signature on the given user challenge. First, the module can extract the key temporarily from the TPM and sign the challenge. Second, given the key is supported by the TPM, the TPM itself can handle the signature creation. The challenge is sent to the TPM to perform the signature creation operation within its security boundary. To complete the identification operation, the user receives the signed challenge and verifies the signature. On success, the module is the correct module.

This enhancement to plain certificates has one minor advantage but also disadvantages. The advantage is that the module's private key is protected by the TPM and therefore cannot be read from outside. The NVRAM solution does not limit algorithm selection but needs key disclosure for key usage. The key has to be loaded to RAM and therefore can be eavesdropped. The solution imports the module's private key into the TPM. As from now, the private key cannot be read by anyone, not even the module itself. Consequently, attacking the module's private key is not possible any more. The main disadvantage of using TPM services is its very limited selection of encryption engines and poor performance compared to software solutions. Regardless of which storage solution is selected, the key has to be shipped with the TvSM instance. Consequently, key extraction is possible before setup and therefore before the TPM comes into play.

All in all, the enhanced certificate solution is more secure than the plain certificate solution discussed in the section above. The main drawback is that an attacker can retrieve the module's private key prior to setup. Drawbacks like lack of performance and narrowed flexibility are outweighed by the security gain. All in all, the solution is not secure enough for module identification.

Take Ownership Procedure

All solutions discussed yet require at least one injected secret per shipped instance. That complicates packaging on the one hand, on the other hand it is a security concern. Some sort of take ownership procedure could make secret injection superfluous and therefore eliminate related drawbacks. A take ownership procedure is used for initializing a TPM for instance.

No secrets or keys are included in the shipped package. During setup, some sort of secret can be imported or created. That can be a password or PIN for secret comparison as well as a private key for certificate-based techniques as described in the preceding sections.

Again, there are advantages and disadvantages. The main advantage is that there is no need for pre-shipment secret injection. This eases up packaging on the one hand. On the other hand, it eliminates eavesdropping the secret directly from package. That significantly improves the overall security level. Another advantage is that one can set up multiple module instances from one shipped package. This allows cluster operation, for example. The main disadvantage reflects in the fact that the responsibility of a secure setup is handed over to the customer. The customer has to make sure that he is talking to the correct module during setup. This is not trivial with the TvSM running in a virtualized operating system on a host hardware. Besides that, the TCG Software Stack needed for TPM communication is capable of TPM access via a network connection. Given the fact that such a network connection can be opened by another (malicious) compartment running on the same host hardware complicates things even more. If the customer tells a compromised module the secrets which are meant for secure module identification, it is hard to detect a rogue module afterwards.

Great flexibility in module setup and more secure handling of secrets while eliminating eavesdropping attacks is what is desired. Nevertheless, during module setup new attack vectors are available. Those attacks have major impact on module security. The time frame for attacks is disproportional smaller than for attacks the take-ownership-procedure eliminates. Nonetheless, it is still possible to impersonate a security module by timing the attack to module setup, for example. Therefore, a plain take-ownership procedure as described here is not suitable for security module identification.

Utilizing the TPMs sealing feature

Better control over a target module being compromised or not is given by taking the module's state into account. That can be done by using the TPMs sealing feature. We discuss a combination of a subset of the certificate solution discussed before and the sealing process in this section.

Given a key pair for module identification, the module can seal its private part of the key to a certain platform state. A precalculated state value shipped with the module, for example, would work well. Given that, the module can unseal and therefore use its private key for identification purpose if and only if the module is in a known good state.

The main advantage of this solution is that a compromised module cannot use its key for identification purpose. Besides the challenges of key distribution and shipment (discussed in the previous sections) another disadvantage shows up. Whenever the module is to receive a software or configuration update, the key has to be resealed to meet the new software configuration. That requires an unsealing process which requires the deprecated configuration of the module and dedicated methods to perform an identification key reseal given the new state. This is a critical process. It allows a denial of service attack by resealing the identification key to an arbitrary value. Further, it complicates handling of the TvSM significantly.

The sealing feature can increase module security well. Still it is not usable as is due to the discussed distribution, shipment and update issues.

Joint Solution

Our presented solution is a joint approach cherry-picking properties from the solutions discussed yet. It includes a take ownership procedure, the public key based authentication part of

certificates as well as TPM features to form a neat and clean solution.

The solution we chose for the TvSM works as follows. We borrow the a public key authentication mechanism from the certificate based solutions discussed above. The TvSM uses a so-called ID key, an RSA key pair, to sign a user-provided challenge. A successful signature verification authenticates the module. Further, to sidestep the key distribution challenge, a take ownership procedure is used. Prior to module setup, the ID key, a non-migratable binding key is created inside the TPM. During setup the key blob of the ID key is supplied to the TvSM. Given that, the module identification calculations can be done within the TPM's protected capabilities on the one hand, on the other hand an attacker cannot get hold of the private part of the ID key because it is protected by the TPM. Therefore, a strong module authentication reached.

The solution has many advantages, but a few disadvantages remain. Thanks to the take ownership procedure no data has to be injected at build time. Also, no media such as printed keys or PINs have to be matched prior to packaging. The take ownership procedure does not disclose any private data. The ID key is generated and used inside the TPM. Given the ID key being not migratable, the key cannot be moved to another machine. That enforces an individual ID key for every module/host. One disadvantage is that a key secret is necessary to operate the ID key. This secret has to be provided to the software by storing it in plaintext. Therefore, a read/write protected storage is needed to shield the secret against unauthorized access. Further, TPM identification is left to the administrator which is critical to a safe and secure take ownership procedure (discussed in a previous section). To seal the ID key usage to a known-good module state is not feasible because resealing is not possible. The acTvSM platform on the contrary encrypts its disks. The key used for encryption is sealed to a known-good platform state. Therefore, the module's ID key is available for module identification if and only if the module is in a known-good state.

The TPM features ensure a high level of protection to sensitive key data as well as impersonation protection. Disadvantages can be fixed using other techniques, but are tolerated to keep usage simple. All in all, the joint solution offers a arguable compromise between usability and security.

Summary

In this section we discussed different approaches suitable for module identification. Starting off with a simple solution using pre-shared secrets and manual comparison we cover more advanced techniques like public key cryptography and signatures. All of these solution are based on some sort of secret, a PIN, a password, a key, which is irrevocable bound to a shipped TvSM instance. Besides that, we discussed the use of a take ownership procedure which deprecates the need for such a fixed secret. The solution is more flexible but also demands for a more complex and critical setup procedure compared to the fixed secret solutions. Our final solution cherry-picks properties from all solutions discussed in this section. It uses a take-ownership procedure which is supported through the TPM and therefore form a good solution to the module identification challenge.

4.4.4. Communication

After module identification, the challenge of getting a secure communication between operator (client) and module arises. There is a variety of attack vectors that the module has to withstand, for example man-in-the-middle attacks, impersonation attacks, replay attacks or others. The main goal in the TvSM context is the protection of keys and secrets against disclosure and abuse. That can be threats like eavesdropping sensible data, replay abuse and denial of key access.

Securing a communication line is a task that can be divided into multiple parts. For an illustration see Figure 4.5. The line in fact starts at the operator, be it the user or the admin. The overall level of security depends on the operator. Of course, no assumptions can be made on how the module will be used. Consequently, the operator has to be assumed the weakest part of the communication. Hence, all other components in the chain of communication should be more secure than the most secure operator imaginable. The next part along the

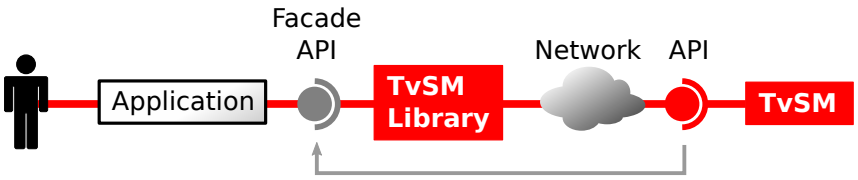


Figure 4.5.: Line of Communication

communication line is the client application. Running at the client host it is the interface between operator and a certain service which may or may not make use of the TvSM. To keep the functional footprint of the TvSM small the client application is defined to be outside the module's responsibility.

In the TvSM context the communication line starts at the client library used by third party applications. The library serves as visible interface between client application and the security module itself. It is therefore a main target of attacks. The interface also hides the inner workings of communication, which is the network part of the communication line as well as the security module itself, which is the second peer of communication.

To meet these requirements we evaluated different protocols: the Object-Independend Authorization Protocol (OIAP)/Object-Specific Authorization Protocol (OSAP) [40], the Transport Layer Security (TLS) protocol [79], as well as the Session Key Authorization Protocol (SKAP) [76].

OIAP/OSAP

Both protocols, the Object-Independend Authorization Protocol (OIAP) and the Object-Specific Authorization Protocol (OSAP), are part of the TCGs specification and therefore used for TPM communication. Both were designed to require a minimum of computational effort and meet requirements like encrypted secret transmission, replay prevention, as well as mutual authentication to name a few. The minimalistic design results in very specific protocols with all their advantages and disadvantages.

Both protocols provide various services for secure communication. Message integrity is provided. Every transmitted data blob is hashed and sent alongside the original data blob.

A session key turns the hash into a HMAC. With that, message authentication is provided. Performing those HMAC calculations on both sides of the communication line leads to mutual authentication without any secret transmission. Further, the rolling-nonce paradigm enables live-checks to prevent replay attacks. This is accomplish by adding a fresh nonce to every new message. The response message includes that nonce as well and a live check can therefore be performed. OIAP is used to secure a communication session over a various number of TPM objects and operations, while OSAP authenticates just one TPM object, one single key for example. OSAP additionally provides encrypted transfer of data which is needed for secret injection. More on the inner workings of the authorization protocols is stated in the TPM specification [40].

These protocols have advantages and disadvantages. One advantage is their minimalistic design. It results in low computational effort for the TPM. That helps keeping the TPM slim and cheap. Another advantage is their specificity. Each protocol is clear and simple for its intended purpose. That promises a small chance of hidden vulnerabilities. Nonetheless, formal methods showed flaws on the OIAP being vulnerable to replay attacks [72, 108]. A great disadvantage is the weakness of the protocols with use of shared secrets [76]. The scenario of having one SRK with a standard secret known to everyone is present in every single use case. Therefore, the flaw is very critical. This scenario is present in the TvSM context also. Another disadvantage from a TvSM's point of view is the necessity of two different protocols for module operations. This doubles the attack surface.

Taken together, OIAP and OSAP are not suitable for use within the TvSM. First, the computational effort is not important. There are much more resources available as there are on a TPM. Further, the fact that the full operation of the TvSM would require two different protocols will result in more protocol-based attack vectors as well as a more complex protocol selection and communication. Besides that, the flaws known for shared secrets and replay attacks are too critical to be ignored.

SSL/TLS

The Transport Layer Security (TLS) protocol [79], former Secure Sockets Layer (SSL) protocol, is a widespread protocol for securing Internet communication. It is a standard and under continuing development.

The protocol provides a network socket to network socket encryption implemented within the application layer. It supports mutual authentication as well as replay detection/protection. The session key is generated with the use of public key cryptography. Further, message authentication and message integrity checks using the HMAC technique are available. For further information the reader is referred to the official standard [79].

TLS has a couple of advantages. First, the whole traffic is encrypted with a symmetric session key. This enables transmission of secrets without any additional cryptographic effort. Further, for client authentication no server-side plaintext secret is necessary. That eases up the module's session secret protection tremendously. The second advantage is that the protocol is well tested and several implementations exist. Therefore, it would save time and effort when selected. A disadvantage of a widespread and complex protocol is the higher chance of yet undetected vulnerabilities as [81] and [103] indicate.

On the whole, the protocol is well suited for securing the TvSM's communication lines.

It has a wide range of security features such as mutual public key authentication, message authentication and replay prevention just to name a few. The downside is the protocols complexity. Therefore, new vulnerabilities are much more likely as for the before mentioned OIAP and OSAP. Nevertheless, the protocol was not used in TvSM design.

SKAP

The Session Key Authorization Protocol (SKAP) is the result of fixing OIAP/OSAP. It is designed to replace OIAP and OSAP for TPM usage. By merging features of TLS to the available protocols SKAP is a more powerful protocol which deprecates the need for multiple protocols for different tasks. As OIAP and OSAP, SKAP is designed to be lightweight to help keeping TPMs small and cheap.

SKAP provides mutual authentication just like its predecessors do. Message integrity checks and live-checks due to rolling nonces are available as well. One difference is that SKAP offers object independent authorization by transforming a successful object authentication to a successful session authentication. Besides that, it is possible to transmit encrypted secrets within an open session. An illustration of the SKAP is given in Figure 4.6. kh reflects a TPM key handle, $PK(\cdot)$ extracts the public key from its argument. ah denotes the authorization handle, n_{even} and n_{odd} are nonces. $E_x(y)$ encrypts y using the key x. x' and x'' denote a newer versions of x. The full inner workings can be found in [76].

One big advantage of SKAP is its ability to extend a successful object authentication to a valid session authentication. In other words, one can start a OSAP-like object specific session and also process other objects within that session. This saves time and computational effort. Advantages and drawbacks of minimalistic protocols were already discussed in the OIAP/OSAP section. A drawback from a TvSM point of view is the need for plaintext secrets on the module's side of the communication line. This requires additional effort to shield the secrets from unauthorized access. This is no issue for TPM usage because the TPM has build-in hardware isolated storage.

Overall, SKAP cannot be used directly in the TvSM design. Object specific authentication does not match the required authentication type discussed in the next chapter. The task of shielding plaintext secrets within the TvSM is done by its underlying acTvSM platform. So that is no problem neither. The ability to inject secrets within an open session perfects a promising protocol.

Modified SKAP

Plain SKAP is not directly compatible with the requirements of object independent authentication like discussed before. A minor modification to the protocols session initiation command resolved the incompatibility.

To meet the required authentication we modified the session initiation command slightly. More precise, the public key handle kh was replaced by the authentication data being the role identification string $role$, the $ad(kh)$ data from the original SKAP by the role identification secret $secret$. An illustration is given in Figure 4.7. The second minor adjustment we did was replacing the key referenced by the key handle pkh by a key of the module $TvSMKey$. Which is the module's ID key for maintenance operation and the module's master key for common operations. See Table 4.1 for an illustration. Authentication methods are discussed later.

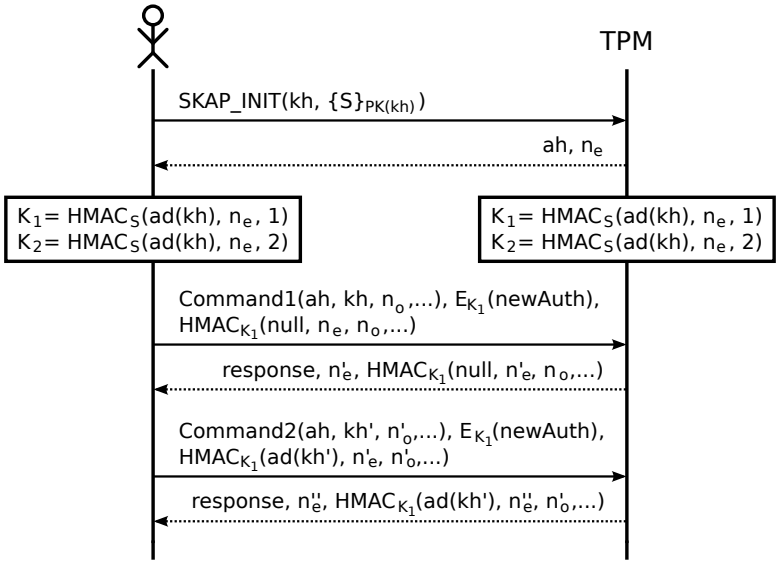


Figure 4.6.: SKAP [76]

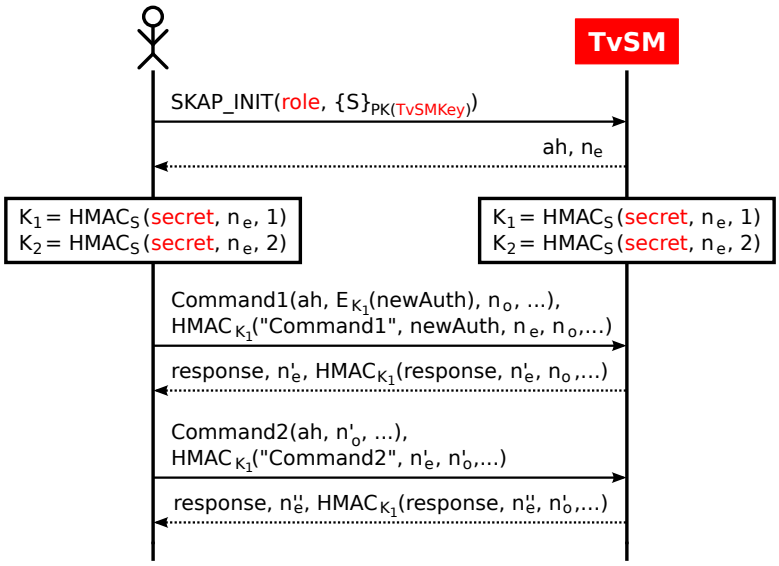


Figure 4.7.: Modified SKAP

To sum this up, the modified version of SKAP meets the module's requirements quite well. Password support for the required authentication mode, secret injection build-in and the desired module identification (see Section 4.4.3) for free makes it a suitable solution.

Summary

In this section three different communication protocols which were evaluated during the TvSM design are discussed. First, Object-Independend Authorization Protocol (OIAP)/Object-Specific Authorization Protocol (OSAP) which are required by the TCG specifications were reviewed. Two different protocols would be necessary for TvSM operation. Further, security issues are known. Given that, we did not use these protocols in our presented design. Next, we evaluated the use of Transport Layer Security (TLS) for our design. Due to its known security issues and its complexity, we did not use this protocol either. After that, we reviewed a new protocol designed to fix the issues of OIAP and OSAP. The protocol is called Session Key Authorization Protocol (SKAP). The protocol does not fit the authentication mechanisms as-is and was therefore not used in our design. Finally, the approach we used for our design is explained in detail. The solution is a slightly modified version of SKAP to fit our authentication mechanisms.

4.4.5. Summary

In this chapter essential design steps for module operation are discussed. That includes role identification as well as a discussion on authentication types. For our presented design we selected two roles, a privileged role for maintenance tasks and a standard role for operation. Different authentication types which work well with the identified roles are covered. Furthermore, module identification and communication between operator and module are discussed. It is explained why module identification is a critical task and how it is solved in our design. Last, different communication protocols are discussed.

5. Security Analysis

In this chapter we give a compressed overview of security concerns. We recapitulate engineering decision which have major impact on the TvSM security as well as a compressed list of remaining attack vectors. This discussion is meant as a security statement as well as a list of open issues for potential future releases.

We discuss different parts of our design starting at the module's setup procedure and ending at daily use cases of the TvSM.

TvSM Identification For module identification we introduced the so-called module Identity Key (ID key). The ID key is a non-migratable binding key managed by the TPM.

Given that, it is hard to attack this key if it already exists. The critical part is the key creation process. It is accomplished during the take-ownership procedure while TvSM setup. First, the key is generated within the TPM providing a key authentication secret. Next, the key blob return by the TPM is handed over to the TvSM setup routine along with the authentication secret. Now, the TvSM instance is bound to the TPM.

Given the virtualized nature of the TvSM, setup has to be controlled within another virtualized operating system. The TCG Software Stack, which is used for TPM communication, is capable of remote TPM access. One may use a operating system for running the setup process which is compromised such that its TCG Software Stack uses a remote TPM. The target TPM can be a (malicious) TPM emulator running in the operating system itself or a TPM of a foreign platform which is reachable via network. This attack vector allows getting control of the TvSM's most critical key. Therefore, it is highly important to be sure which TPM is used for ID key creation.

Server Protection The TvSM core executable is protected by the acTvSM platform. The acTvSM platform protects the core executable in power-down state, during the boot process and while operational. Every compartment image is encrypted when stored on disk. Therefore, the image data is protected against modifications. The secure boot feature of the acTvSM platform guarantees that the TvSM is successfully started up if and only if the software state is trusted. In online mode, the resource isolation feature of the acTvSM platform forecloses read and write access from outside of the compartment.

The acTvSM platform provides an update mode as well. As said before, software modifications cause a state change. The update mode allows software modification with a subsequent adaption of the new trusted state values. The update mode therefore allows undetectable software modification. More critical is the fact that resource isolation does not hold for the privileged user operating the update mode. The ID key usage secret is stored on disk in plain text. One can read the secret and use the corresponding key, the ID key, to disclose the master key.

Communication Communication between the client library and the TvSM core executable is protected by the SKAP protocol. The protocol guarantees liveness of communication, secrets are blinded prior to transmission and mutual authentication guarantees the identity of both sides. Liveness is realized through a nonce handshake and secrets are encrypted with the session key. The session key is derived from a one-time-pad supplied by the client. Mutual authentication is accomplished by encrypting the one-time-pad with the module's public ID key. That way, the session keys only match if both sides use the correct keys for session key creation. On key mismatch, secrets cannot be decrypted properly. All in all, there are no attack vectors left concerning the protocol.

We designed the API to be narrow and lightweight. The API should give a small attack surface to foreclose various API-level attacks. Moreover, formal methods are going to be used for testing the API. No results are available yet.

Key Management Key blobs are protected by a cryptographic checksum which prevents a modified blob to be used in cryptographic operations. Each modification, be it modification of the checksum, the key data or the key attributes, invalidates the key. The external storage solution accomplished through key wrapping prevents from disclosure of key data while stored on world readable media. Key slot lifetime is directly connected to the session lifetime and therefore to the session timeout. That guarantees that loaded keys are no more usable at session end. This complicates extensive session hijacking. Session hijacking is possible but requires the session state which includes the session secret to be known. The session secret is held inRAM of the client host and retrieval is therefore difficult.

Services All services of the TvSM are protected by proper authentication secrets. For signature creation, for example, one has to provide the signing key's secret. Key migration demands for the key's secret, the secret used for authenticating the privileged role or a combination of both.

We chose conservative key import/export constraints. It is not allowed to import or export a storage key. Therefore, disclosure of parts of the key hierarchy is foreclosed. Moreover, only imported keys can be exported again. Therefore, keys which were created inside the TvSM remain protected throughout their lifetime.

Key migration is accomplished through a re-wrap-operation within the TvSM. Prior to the operation, the new parent key is checked to be a valid TvSM storage key. Due to the fact that storage keys cannot be imported or exported, disclosure of the succeeding key hierarchy is foreclosed.

Attack vectors exist for the time stamping service. After the clock being checked and approved by the admin, the time passed on the host system clock is compared with the time passed in accordance with the TPM tick counter. Whenever the divergence exceeds a configurable threshold, the clock mode is annulled to normal operation and the time stamping service is disabled. The time check is performed periodically with a configurable interval. Due to the fact that the interval is greater than zero, a certain amount of time is available to perform an attack on the time stamping service. An attacker can change the clock and time stamp a document before the next check is performed. We propose a time check prior to time stamp creation to fix this vulnerability.

6. Performance Measurements

To get an idea on the target performance the TvSM should offer to compete well with the products of the market we ran a set of tests. These tests include a performance comparison of available security modules as well as a test on the performance of TPMs. The performance values reached by the security modules will allow a rough estimation whether or not the TvSM approach will meet its promises. The TPM test will reflect the performance of one of the tools available for TvSM design.

First, the comparison of a subset of the security modules within the market analysis is covered. Then, the TPM test is discussed. The test of our presented TvSM implementation concludes this chapter.

6.1. Comparison of hardware security modules

There are different SMs with different performance levels available. We did a representative comparison. The results for RSA performance of HSMs are given in Figure 6.1.

Lacking reviewed benchmark data, we took performance values from product brochures and third party presentations. As a representative of software cryptographic libraries we benchmarked the SIC Crypto Toolkit (highlighted red in Figure 6.1). Our setup was a Linux kernel 2.6.31-21 and Java JRE 1.6.0.20 on an Intel Core 2 Duo P9500 CPU running at 2.54 GHz per core in 64 bit mode. The Crypto Toolkit uses only one core for signature creation. We took ten measurements and averaged them to form the final performance value.

The slowest HSM with just four signatures per second is a cryptographic SmartCard intended to be used as a portable personal devices for signature creation and encryption. The fastest device is intended to accelerate cryptographic services and performs 13 000 signatures per second. The greater part of the compared security modules range between 100 and 1 200 signatures per second. The SSM SIC Crypto Toolkit achieves a total of 5.500 signatures per second.

Generally, high security modules like the Ultimaco CryptoServer 2000, the ARX PrivateServer, or the AEP K1200 HSM show a significant lower performance than other evaluated modules, which might be based on their optimization for security and not for performance. Further, the software solution offers a good performance in comparison to the majority of the evaluated HSMs.

6.2. Benchmarking TPMs

For a better understanding of the capabilities provided by a common TPM we took a series of measurements. The goal was to get an idea of a TPMs performance accomplishing different tasks such as key creation, signature creation as well as hashing. Performance evaluation of

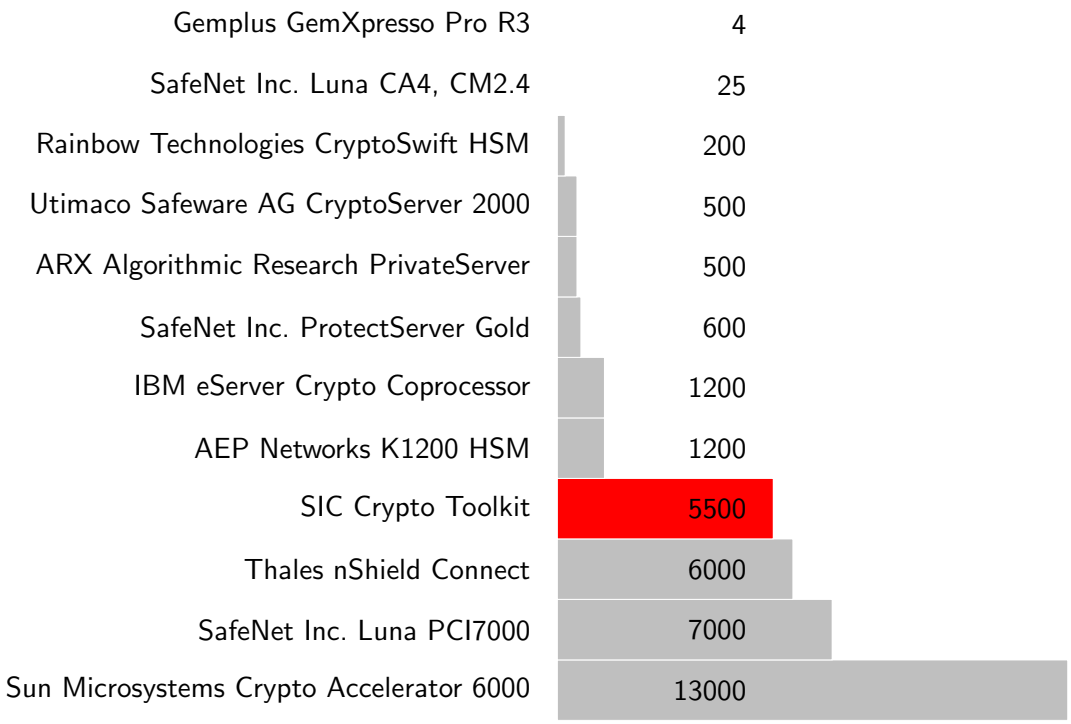


Figure 6.1.: Performance comparison of commercial security modules: RSA 1024 bit signature creation operations per second. Information is taken from product briefs. SIC Crypto Toolkit is a measurement and the only software security module in the comparison.

plain TPM commands were done by [119] already, but they did not take the context, being Java and jTSS, into account.

The TPMs which we evaluated are listed in Table 6.1. The setup was a Linux operating system, jTSS [12] as library and a short Java application which triggered the operations and

Table 6.1.: Reviewed TPMs

Vendor	**Version**	**Abbreviation**
Atmel	1.2.13.9	ATML
Broadcom	1.2.6.77	BRCM
Infineon	1.2.1.2	IFX
Infineon	1.2.3.16	IFX 2
Intel	1.2.5.2	INTC
ST Microelectronics	1.2.8.16	STM

tracked the execution time. Fifteen measurements were taken each and averaged to form the final results. Full information is given in Table 6.2 and Table 6.3.

The results for key creation, and signature creation are illustrated in Figure 6.3, and Figure 6.4 respectively.

For key creation, we observed a high variance in execution time. That hints to true RNG (discussed in Section 2.1.1) used for random key generation. For creation of 2048 Bit RSA key pairs it seems that some TPMs use some sort of precomputation mechanism. The

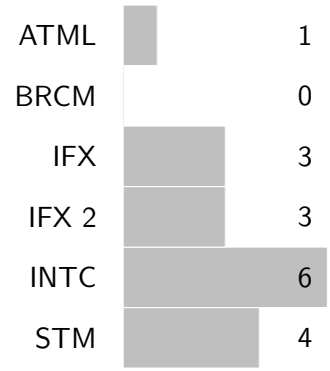


Figure 6.2.: Number of precomputed RSA 2048 Bit key pairs for each TPM

measurements showed that the first few generation operations are completed very quickly and in constant time. The remaining measurements took longer. The number of apparently precomputed keys for each evaluated TPM is illustrated in Figure 6.2.

Fastest creation of 512 and 1024 Bit RSA keys is offered by the Broadcom TPM with about half a second and little less than two seconds respectively. For 2048 Bit keys the fastest TPM is Intel's TPM, which needs an average time of about 7.5s. This result correlates to the number of precomputed keys available. The slowest TPMs for 512, 1024, and 2048 Bit are Atmel,

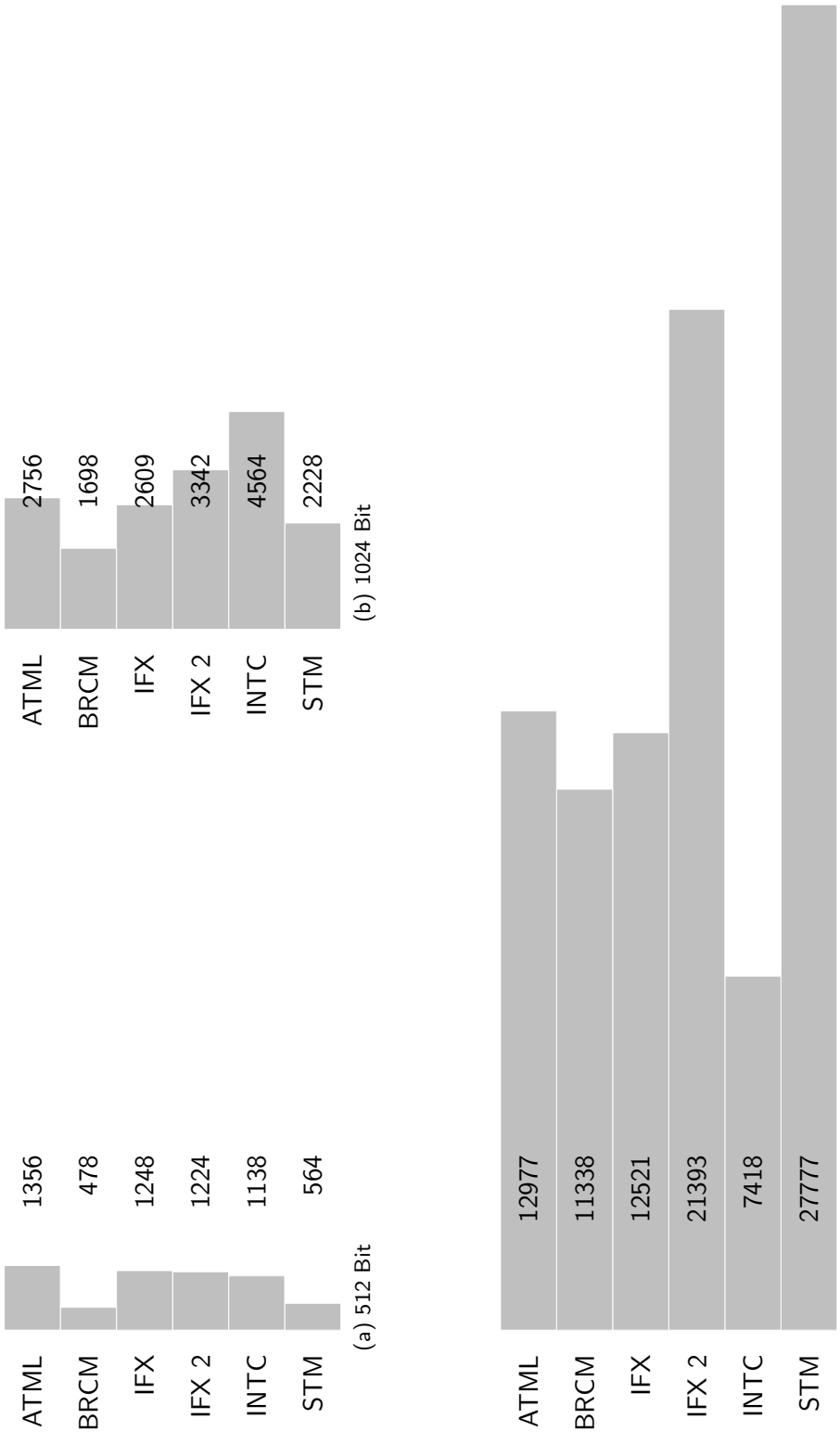
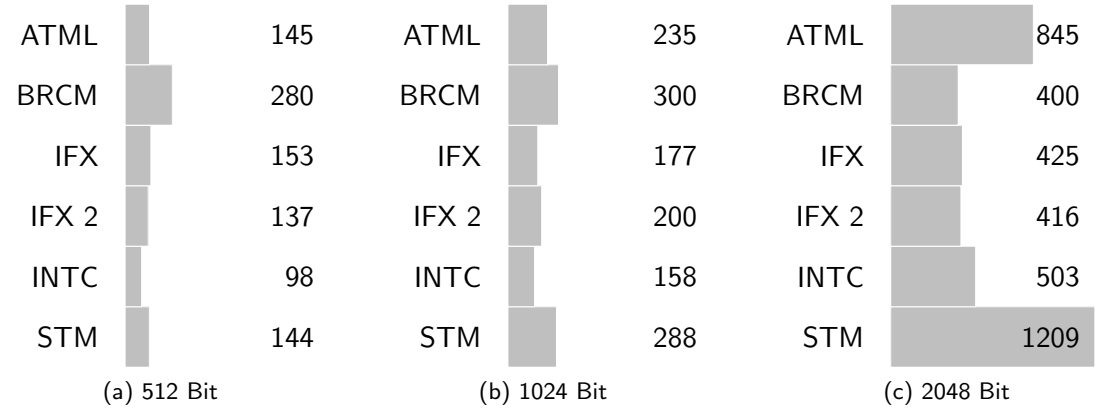


Figure 6.3.: TPM key creation operation mean duration in milliseconds

Intel and STMicrosystems with 1.2s, 4.5s, and about half a minute respectively. At large, our results represent one single test run. Another run might result in values which clearly differ from the values stated in Figure 6.3. Nonetheless, a creation time of half a minute (worst case) for a single 2048 Bit RSA key shows that a TPM is not suited for massive key creation.

For signature creation, a small variance in execution time was observed. The fastest TPM for creation of 512 Bit and 1024 Bit RSA key pairs is the Intel TPM which accomplishes the task in about 100 ms and 150 ms, respectively. A signature creation using an 2048 Bit RSA key is done in about 400 ms by the Broadcom TPM. The worst performance for signature



(a) 512 Bit (b) 1024 Bit (c) 2048 Bit

Figure 6.4.: TPM signature creation operation mean duration in milliseconds

creation using 512, 1024, and 2048 Bit RSA keys is achieved by Broadcom, Broadcom and STMicrosystems with about three, three and less than one signature creation operations per second.

These results show that a TPM is not suited for massive signing operations neither.

All TPM throughput is limited by the communication line, namely the Intel® Low Pin Count (LPC) bus. The LPC bus is capable of a throughput of about 2.5 MB/s [24]. The performance of hashing of data blobs is therefore limited to a maximum of 2.5 MB/s and can therefore not compete with a CPU, for example.

All in all, the measurements show that a TPM is not designed to do any kind of mass service, be it key creation, signature creation or hashing. The TPM is designed to offer cryptographic service at a similar level as cryptographic SmartCards do. In this area of operation performance is not important, cost is.

6.3. TvSM performance

Even though the main goal of the project was to design a security module which is low in cost and high in security, we ran a performance test on it. In this section, the results of the performance tests are presented. They give a rough idea on how well our presented TvSM implementation competes with the commercial security modules compared in Section 6.1.

The TvSM server as well as the client application were operated on a platform with a Intel Core 2 Duo P9500 CPU running at 2.54 GHz per core in 64 bit mode running a Linux kernel 2.6.35-22 and Java JRE 1.6.0.22. 100 measurements were taken each. The client RNG

TPM	512 Bit				1024 Bit				2048 Bit				TPM
	min	max	mean	σ	min	max	mean	σ	min	max	mean	σ	
ATML	185	2117	1356	420	186	6700	2756	1642	198	54881	12977	15482	ATML
BRCM	339	993	478	181	538	3538	1698	821	4855	26063	11338	7170	BRCM
IFX	568	2639	1248	509	1081	11281	2609	2541	809	40465	12521	14055	IFX
IFX 2	536	3240	1224	904	905	7773	3342	1549	805	43021	21393	14255	IFX 2
INTC	588	2103	1138	420	1901	15058	4564	3215	320	59570	7418	16134	INTC
STM	491	1190	564	223	1020	8308	2228	1838	365	84852	27777	24547	STM

Table 6.2.: TPM key creation operation performance measurements [ms]

TPM	512 Bit				1024 Bit				2048 Bit				TPM
	min	max	mean	σ	min	max	mean	σ	min	max	mean	σ	
ATML	144	148	145	1	227	239	235	2	836	859	845	6	ATML
BRCM	259	300	280	14	260	320	300	17	380	422	400	16	BRCM
IFX	136	161	153	6	160	186	177	10	416	436	425	7	IFX
IFX 2	133	156	137	6	196	209	200	3	408	421	416	3	IFX 2
INTC	96	100	98	2	157	161	158	1	498	504	503	2	INTC
STM	142	148	144	2	276	293	288	5	1201	1218	1209	5	STM

Table 6.3.: TPM signature creation operation performance measurements [ms]

initialization procedure was excluded from the measurement. It lasts for about one second when using the `java.security.SecureRandom` RNG shipped with Javas Java Cryptographic Extension (JCE). This procedure is done prior to the clients first run of the RNG and therefore only happens once during an applications run time.

The results are listed in Table 6.4. The session init process lasts for about 750 ms. Creating

Operation	**Performance [ms]**			
	min	max	mean	σ
session init	708,0	814,0	766,0	25,8
create key	21,0	378,0	62,0	60,8
load key	12,0	21,0	13,0	1,1
sign	6,0	9,0	7,0	0,6
session shutdown	1,0	11,0	2,0	1,1

Table 6.4.: TvSM performance (RSA 1024 bit) from the clients point of view

a 1024 bit RSA key pair takes a median time of about 60 ms. It takes about 15 ms to load a key into the TvSM. With a time of 7 ms per operation, the TvSM is capable of creating about 150 signatures per second using a 1024 bit RSA key pair. Session shutdown is done in about 2 ms.

In our proof-of-concept implementation, the protocol overhead has major impact on the performance of the TvSM. The use of real parallelism available on modern CPUs as well as a performance-optimized implementation would cause a major improvement. For example, a CPU operating eight cores with hyper-threading is capable of running sixteen concurrent threads. Using six of these threads for management services for example, ten real parallel threads are still available. Given that estimation, performance can be boosted to a multiple of ten. That would result in a performance of 1 500 signatures per second using a 1024 bit RSA key pair.

6.4. Summary

In this chapter the results of the performance evaluations we made were presented. That was a comparison of RSA signature creation for available security modules and the software crypto library running on commodity hardware, the SIC Crypto Toolkit. The comparison shows that the SIC Crypto Toolkit, which is to be used in our presented TvSM, competes quite well with the other evaluated modules. The TPM evaluation shows that a TPM is not suited for providing high performance services. Finally, our presented implementation of the TvSM was tested. Taking the proof-of-concept characteristics of our implementation into account, it performs quite well.

7. Implementation

In this chapter, we give a brief description of the software design of the TvSM core executable. More precise, only the most interesting parts are picked for description here. To keep the focus on the important parts, we state only simplified diagrams in this chapter. Our presented TvSM implementation is a proof-of-concept implementation. Also, we point out possible enhancements.

First, we discuss the environment and the technologies used. Then, we give an architectural overview. Next, we state primitives of the TvSM software design which form the basis for the common functionalities and the server functionalities.

7.1. Environment

For the TvSM core executable we have chosen the Java programming language version 1.6.0_22. There are multiple reasons for this decision. First, being in production state, the language is well tested, maintained, and various libraries exist. One important library for our implementation is the Java Cryptographic Architecture (JCA)/JCE. It allows simple use of cryptographic services such as encryption or signature creation. The JCA also supports exchangeable cryptographic service providers which is the second important point in our context. For our purpose, we used the SIC Crypto Toolkit [9] as main service provider. For the client library, the standard provider shipped with the JCA was sufficient. That eliminates the need for external libraries for client operation. Another important point for TvSM implementation was easy TPM access. The Java TCG Software Stack (jTSS) [12], which is Java also, was available and we used it for TPM access. Further, Javas Remote Method Invocation (RMI) allows a rather simple network communication.

7.2. Architectural Overview

To keep the software engineering process as well as the outcome neat and clear we split our software design into smaller components. We created four functional components: the server, the client, a common resource and the API, as well as a small component holding different tools. An illustration is given in Figure 7.1. The API forms the interface for the client and the server. The common component is used by the client and the server. In the following, these five components are described briefly.

The Application Programming Interface (API) reflects the TvSM's role layout discussed in Section 4.4.1 as well as the common interface for accessing the client and the server component. Furthermore, common exceptions are defined within the component.

The client component contains the client library necessary for communicating with the TvSM server component. It implements the interfaces defined in the API component and uses common functionality grouped within the common component. The interfaces hide the inner

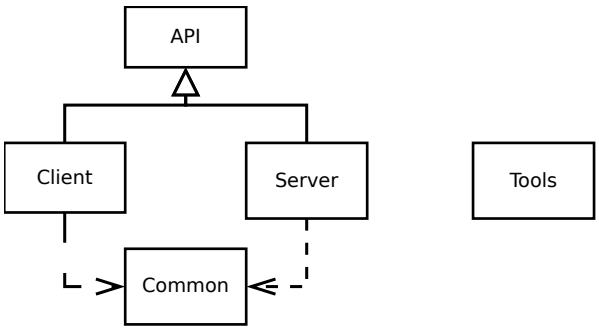


Figure 7.1.: Java Packages

workings of the communication from the user application. For using the cryptographic services of the TvSM, a user application has to have the API component, the client component and the common component available.

The server component contains the core implementation of the services provided by the TvSM. That is a key store implementation for key slot functionality, an engine for cryptographic operations, a timer, a logger, a statistics collection and other services. The server component also implements the interfaces defined in the API component. Common functionality is used from the common component.

The common component contains a set of common resources such as common protocol code, settings handling, primitive implementations as well as utilities like key converter and other tools.

The tools component contains the ID key creator tool necessary for the take ownership procedure prior to module setup. The component has no dependencies on other components and is not used by any other component.

7.3. Application Programming Interface

After a first API layout we decided to use an identical API for client and server. The decision promised a simple implementation without the need to design, implement, and verify another API. The communication line is illustrated in Figure 4.5.

The presented API is illustrated in Figure 7.2. Role distinguishing is done through different subtypes to the basic `Session` interface. Subtypes are the `MaintentanceSession` interface which offers the functionality adequate to the privileged role and the `OperatingSession` interface which offers functionality available in the standard role (more one roles see Section 4.4.1).

Contrary to expectations, the decision of using one API for server and client access complicates things; for implementation as well as for the client library user. To get the design up and working we had to create inheritance models for authentication data containers as well as for results. They are illustrated in Figure 7.4 and Figure 7.5 respectively. This solution works quite well but without type safety at build or run time.

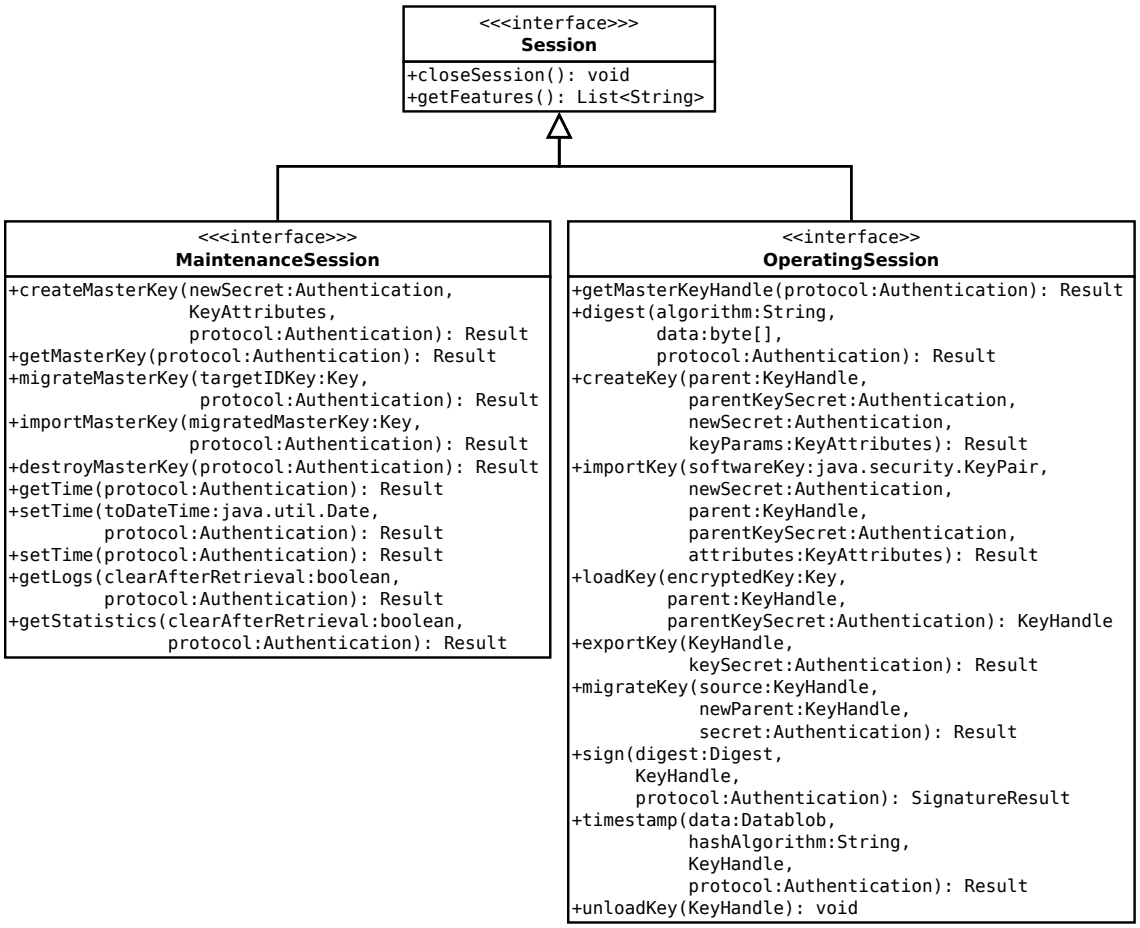


Figure 7.2.: API

7.4. Primitives

In our presented TvSM implementation we use a couple of primitives such as basic data structures. The primitives we discuss here are the `Datablob` object, the interface `Authentication` and its subtypes, the interface `Result` and its subtypes as well as the basic key blob class.

7.4.1. Datablob

We created the `Datablob` interface to provide a common type for arbitrary data. The main application of the interface is to hold binary key parts. Behind the interface, we created two different implementations, a raw data blob type and a wrapped type. The class layout is illustrated in Figure 7.3. The raw type contains raw data. A key, for example, is stored world readable. The wrapped type contains a wrapped blob of data. A key, for example, is stored blinded. This is done through the following procedure. Whenever a `RawDatablob` object is to

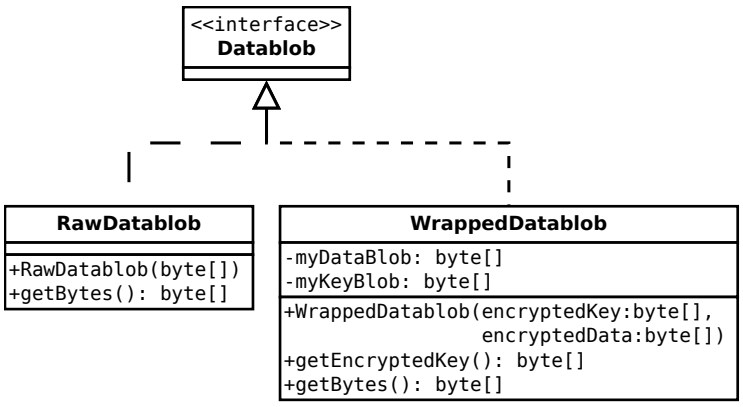


Figure 7.3.: Classes: Datablob

be wrapped, the object is transformed to a `WrappedDatablob` object. That is done by creating a symmetric encryption key. In our solution we use AES with a configurable key size. The key is used to encrypt and therefore protect the raw data given with the `RawDatablob` object. Then, the symmetric key is encrypted by a given public key. Both results, the encrypted symmetric key and the encrypted data blob are stored within the newly created `WrappedDatablob` object. That allows decryption, given the correct private key.

7.4.2. Authentication

The `Authentication` interface holds any kind of authentication data as a `Datablob` object. The interface and its inheritance structure is illustrated in Figure 7.4. A secret as well as a user name used for role identification are the basic types required by our TvSM implementation. To meet the single API engineering decision (see Section 7.3) further subtypes are necessary. One example is the `NoAuthentication` subtype. It is used whenever the interface demands for authentication data but not the operation. Other examples are SKAP related types. They are used to forward the protocol through the common API. The subtypes used for this purpose are the various protocol request types. The use of these is illustrated in Section 7.5.2.

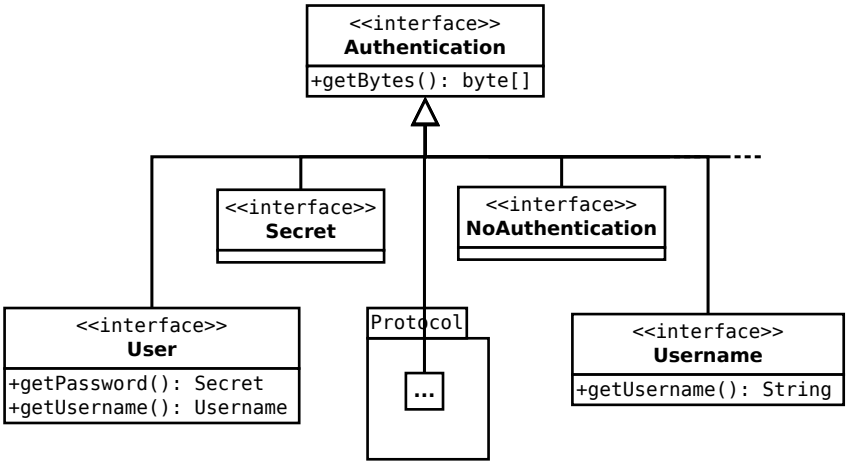


Figure 7.4.: Classes: `Authentication` type and subtypes

7.4.3. Result

The `Result` interface reflects all kind of results created by our TvSM. The actual data is stored within a `Result` subtype as a `Datablob` object. An illustration of the interface and its subtypes is given in Figure 7.5. `Datablob`, `Digest`, and `KeyHandle` are the most intuitive subtypes used. To meet the single API engineering decision (see Section 7.3) further subtypes are necessary. Again, the protocol has to be forwarded through the common API.

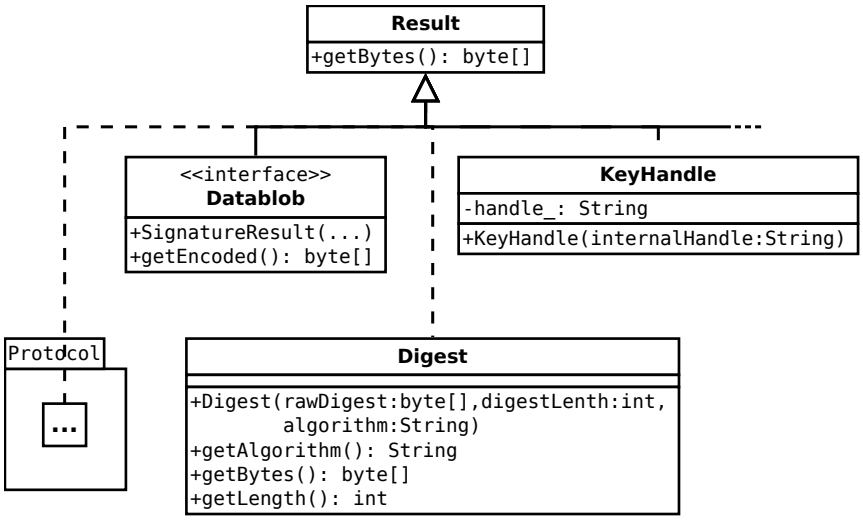


Figure 7.5.: Classes: `Result` type and subtypes

To reach this, various protocol response subtypes were created. The use of these is illustrated in Section 7.5.2.

7.4.4. Keys

The `Key` interface is the basic interface used in all key handling operations. An overview is given in Figure 7.6.

Key attributes, data blobs holding the key parts, key parameters, as well as the key secret are provided. The key attributes are stated by the `KeyAttribute` type. After initialization at key creation the attributes are read-only. Various flags like key type and key length are held. The key parts are held by `Datablob` objects. That allows simple key wrapping by transforming the raw data blobs to wrapped ones. Wrapping is done prior to key export. Unwrapping is done during the key load procedure. The key parameters, the RSA modulus for example, are stored in a keyed list. That results in a flexible parameter handling suitable for various algorithms. The key secret, an `Authorization` subtype, holds secret data in a `Datablob` object. Encryption and decryption (wrapping and unwrapping) is done as it is done with the private key.

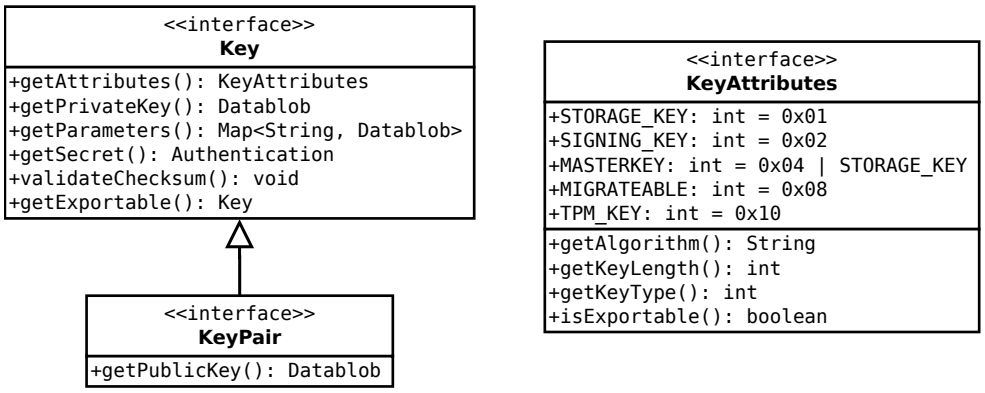


Figure 7.6.: Classes: `Key` type and subtypes

Key protection is done via cryptographic checksums. The principle of key protection was discussed in Section 4.1.2. Each key instance holds a checksum which is set at key creation. Whenever the `validateChecksum` method is called, the checksum is recalculated and compared to the checksum set at key creation. On mismatch, an exception alerts our TvSM about the damaged key. The TvSM then takes further steps like informing the user and error recovery by throwing an adequate exception with a subsequent session shutdown.

The common component (see Section 7.2 for an architectural overview) offers a few key handling services. That is storing and loading a (wrapped) key object to and from disk as well as providing a neat interface to JCE compatible key formats through key converters.

7.4.5. Summary

In this section we discussed the basic building blocks of our presented TvSM. They are used heavily and therefore form the basis of the TvSM. That was the data blob, the authentication and result inheritance models as well as the key object.

7.5. Common Functionality

In this section we discuss the most important services available for the client library and the TvSM server. That is the use of Javas RMI, a short word on the implementation of the modified SKAP discussed in Section 4.4.4 and the settings solution we used.

7.5.1. RMI

For implementation of a network reachable TvSM we used a dedicated tool available in Java. Javas Remote Method Invocation (RMI) allows remote method calls with little implementation effort. There are just a few configurations which have to be done to get RMI up and running.

A registry service is necessary on the server machine. RMI enabled objects can be registered there. The methods of these objects are then available for remote invocation. Remote invocation is done by the client. The client retrieves a list of object available in the servers registry. Given that, he can select a certain object and use its methods remotely. There are basically two options for RMI transmission. That is the dedicated RMI protocol or a tunneled version using the Hypertext Transfer Protocol (HTTP). For our implementation we used the tunneled version to remain firewall friendly.

In the TvSM context we have to distinguish between a privileged role and a standard role which are reflected by two different session interfaces. The modified SKAP requires an authorization handle to be attached every time a message is transmitted (see Section 4.4.4 for further information on the modified version of SKAP). In our presented TvSM implementation we defined the session object to serve as the authorization handle. That requires a dynamic session objects management in the RMI registry.

In fact, we need a remote object management process including creation, delivery and deletion of session objects. A simplified illustration of this remote object invocation process is given in Figure 7.7. First, the standard RMI setup has to be done which is creating a registry. Further, a `SessionFactory` is created and registered for remote use. Then, the client retrieves the object list from the server and asks the remote `SessionFactory` to create a new session. The remote factory creates the session, makes it available for remote use, and returns the authorization handle. The authorization handle which is a random string of fixed length is used for retrieving the session object and therefore make it available for the client. This procedure is done during client side session object creation and is therefore invisible for the user. On session close the server side session unregisters itself from the RMI registry. Additionally, the session has to invalidate itself because the entry in the registry is not removed until no references are active anymore.

7.5.2. Modified SKAP

The implementation of the modified SKAP was straightforward (see Section 4.4.4). An illustration is given in Figure 7.8. Whenever the client initiates a session by creating a client-side session object, an SKAP session init is performed. The provided `SessionInitRequest` is transformed into a `PackedInitRequest`. The latter, which is transfered to the TvSM server, does not contain the session secret. The server then performs the role authentication process, the session creation process discussed in the section before, and other tasks. Then a `SessionInitResponse` is assembled and sent back to the requester. After decoding, session

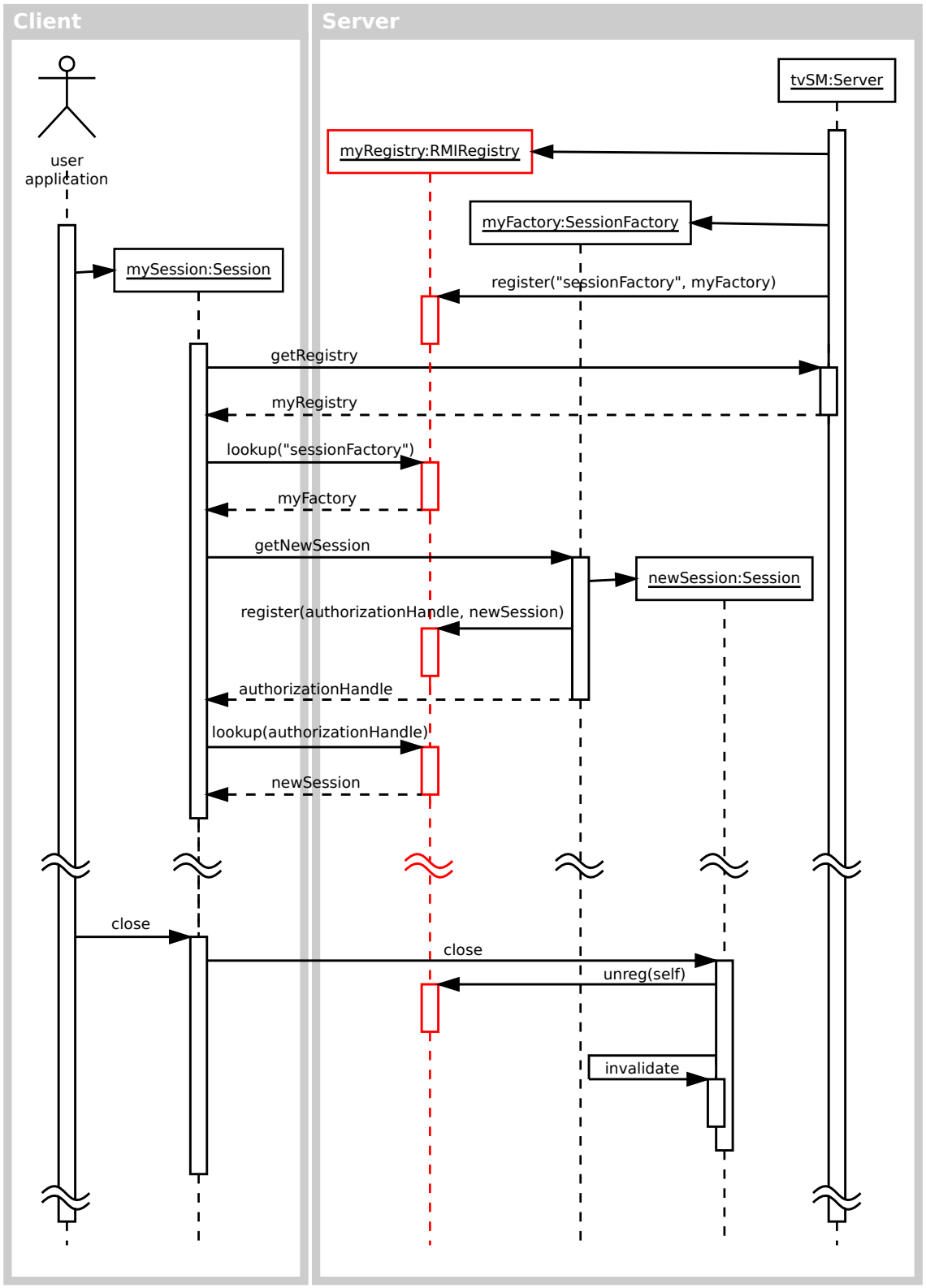


Figure 7.7.: RMI

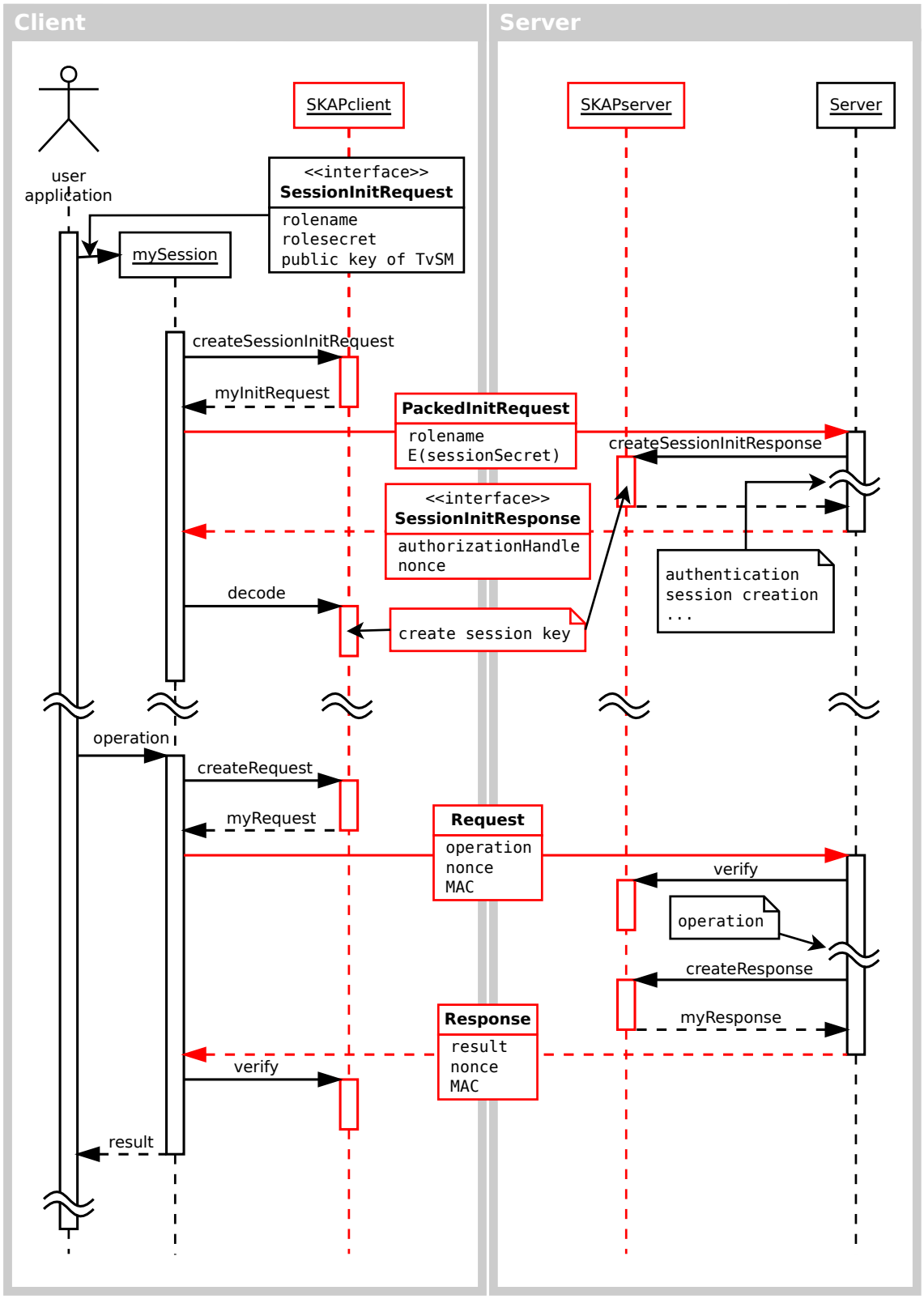


Figure 7.8.: SKAP implementation

key creation, and session retrieval through RMI the session is ready for usage. After that, any operation triggered by the client goes through the same procedure. First, a request is created. The request then contains any protocol-relevant data such as a fresh nonce and a Message Authentication Code. The request is transmitted to the server. The server verifies the request. On success, the requested operation is performed and a response is assembled. The response then contains all protocol-relevant data such as a fresh nonce and a Message Authentication Code as well. To complete the handshake, the client library verifies the response. On success, the result is returned to the user application. Every failure, be it an unsuccessful verification or a session timeout results in session shutdown. All operations listed here are hidden behind the common API.

7.5.3. Settings

To achieve a customizable behavior, we added settings files for the client library and the server. Common properties configurable via the settings files are key lengths and algorithm types. The server additionally provides role name properties, directory properties for key locations and outputs like logs and statistics as well as session limit and timeout properties, and many more.

The settings files are managed by the singleton `Settings` type. Property names are provided by the `ClientSettings` as well as the `ServerSettings` type. An overview of the classes used is given in Figure 7.9. We implemented the functionality using Java's native `Properties` type.

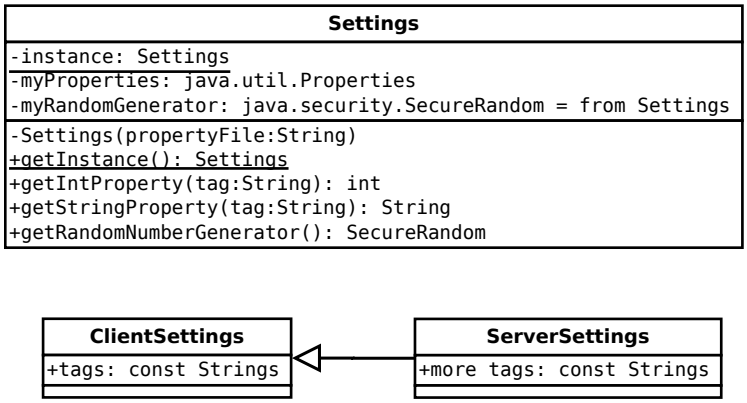


Figure 7.9.: Classes: Settings

7.5.4. Summary

In this section we discussed functionalities used by the client and the server. That was Java's RMI and its use in our presented TvSM as well as the implementation of the modified SKAP discussed in Section 4.4.4 and the configuration solution using settings files.

7.6. Server

In the following, we describe the most important parts of the TvSM software design. They are used exclusively by the TvSM server. That is the key store structure implementing the key slot idea, the scheduler solution reflecting the task based architecture of the TvSM, the crypto engine implementation offering the cryptographic services available, the structures and methods used for reaching the time stamping solution discussed in Section 4.2.4, and the statistics collection. Note that the TvSM server is only accessible through its API. Therefore, secrets can be and are handled in plain text.

7.6.1. Keystore

For using keys within the presented TvSM a functionality we need to store keys for a limited time. Therefore, we introduced the `KeyStore` interface. The interface allows registering and unregistering of keys. First, a key integrity check is done during the register process. On failure, the key is not loaded and the active session is closed. On success, a key is tagged with a key handle.

In the presented design, we use different subtypes. The class context is illustrated in Figure 7.10. That is a non-persistent store, a file-based store and a TPM based key store. The non-persistent key store is used for daily work. A new `NonpersistentKeyStore` is attached to every fresh `OperatingSession` and limited to a configurable amount of key slots. Every key the user loads into the TvSM is stored in a key slot of the non persistent key store. A key handle is created and handed over to the user. The key handle can now be used as a key selection statement in service requests.

The `FileKeyStore` allows key storage in files. A directory has to be supplied during object creation. The directory is used for storing the registered keys. Again, an integrity check is performed to keys during the register process. Here, a key handle is either supplied prior to registering or a key handle is created. That allows static key handles. In our solution, static key handles are used for accessing the modules ID and master key.

Last, the `TPMKeyStore` is a wrapper for the jTSS key store and therefore for the TPM key slot mechanisms. Integrity checks and handle creation is done by the TPM itself. The TPM key handles are mapped to `KeyStore` compatible aliases. This key store was initially intended for managing TPM keys in the module's key hierarchy. In our presented design there is only one TPM key remaining, the module's ID key. Every time the module fires up it loads the ID key from disk using the aforementioned `FileKeyStore` and registers it in the `TPMKeyStore`, which allows usage of the encapsulated TPM key.

7.6.2. Scheduler

Real multi threading, parallel execution of code, gets more and more important. Therefore, we chose a task based architecture for our TvSM design. A `Scheduler` interface allows task execution. An illustration of the scheduler and its context is given in Figure 7.11. Tasks are the wrapping and unwrapping procedures of data blobs as well as the tasks of the crypto engine discussed in the next section. In our proof-of-concept solution, a scheduler offering single-threaded execution is implemented as a default. A scheduler supporting multi-threaded execution can be added easily.

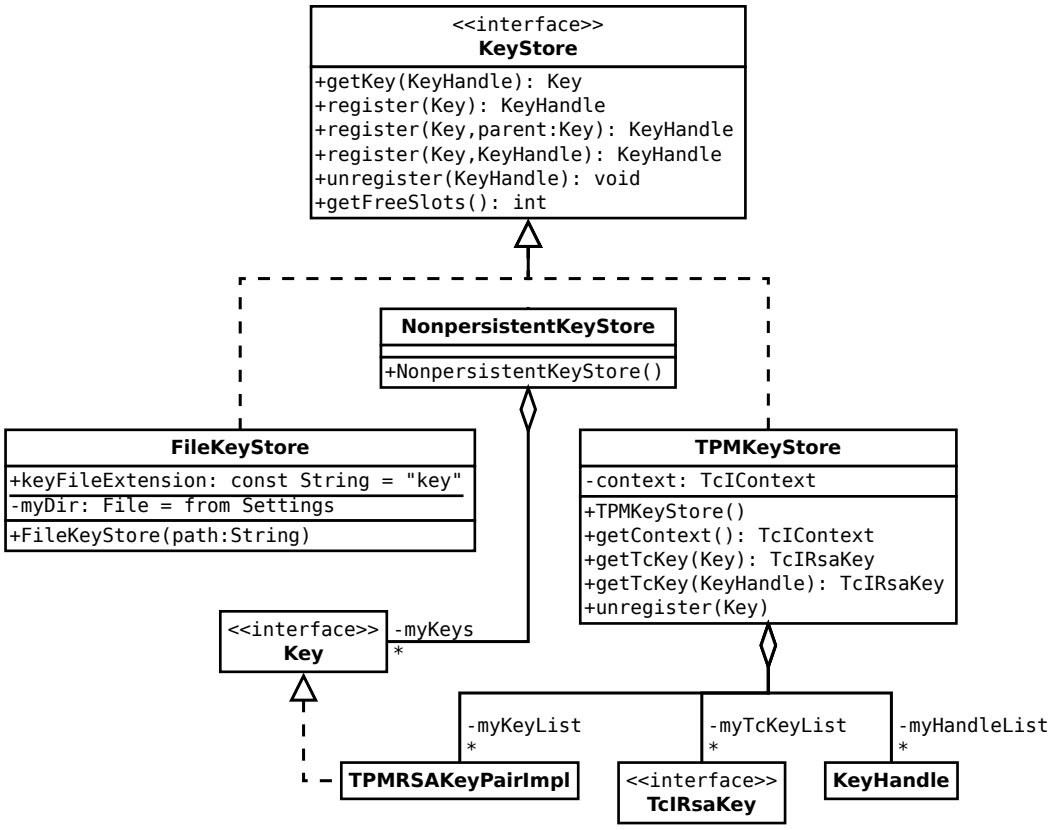


Figure 7.10.: Classes: Keystore

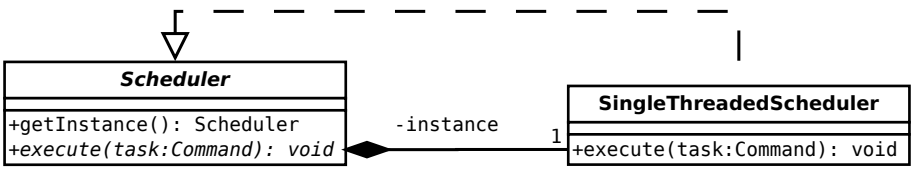


Figure 7.11.: Classes: Scheduler

7.6.3. Crypto Engine

The goal of the cryptographic engine we created for our TvSM was a highly modular and easily extensible collection of cryptographic services of any type. Cryptographic services are encapsulated in tasks which are executed through the aforementioned scheduler. The class layout containing a representative RSA encryption task is illustrated in Figure 7.12. Adding

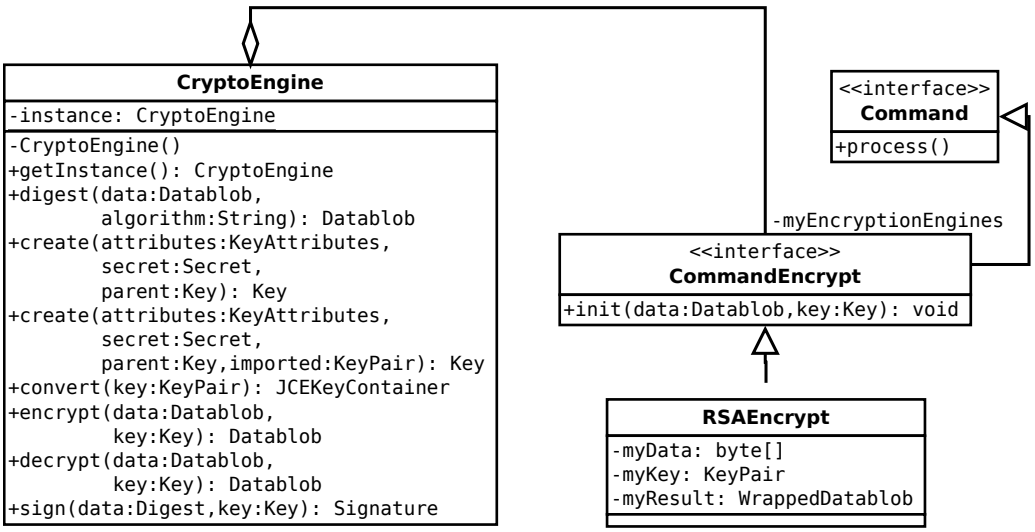


Figure 7.12.: Classes: Cryptoengine

another encryption task is as easy as implementing another subtype of the `CommandEncrypt` subtype and registering it in the singleton `CryptoEngine`.

The `CryptoEngine` class offers a uniform interface for services like hashing, key creation, key conversion, encryption/decryption and signature creation. Furthermore, the engine checks various constraints prior to task execution. The appropriate task (algorithm) is selected using the type of the supplied key. All crypto-related names follow the Java JCE naming conventions [111].

7.6.4. Timer and Time Continuity Check

Due to the time stamping solution presented for our design (see Section 4.2.4) reading only the system clock is not sufficient. Therefore, we introduced the `Timer` class and the `TimeContinuityChecker` class.

The `Timer` class is a singleton class which allows setting and retrieving time information from the system clock. Furthermore, the class manages the TvSM timer mode, which can be approved or non-approved.

Whenever the timer is set to approved mode, the `TimeContinutyChecker` thread is fired up. The thread compares the amount of time passed on either clock, the system clock and the TPM tick count, periodically. The system clock is used for the checking period and reflects therefore the system clock itself. The period and the tolerance can be configured with the corresponding properties in the settings file.

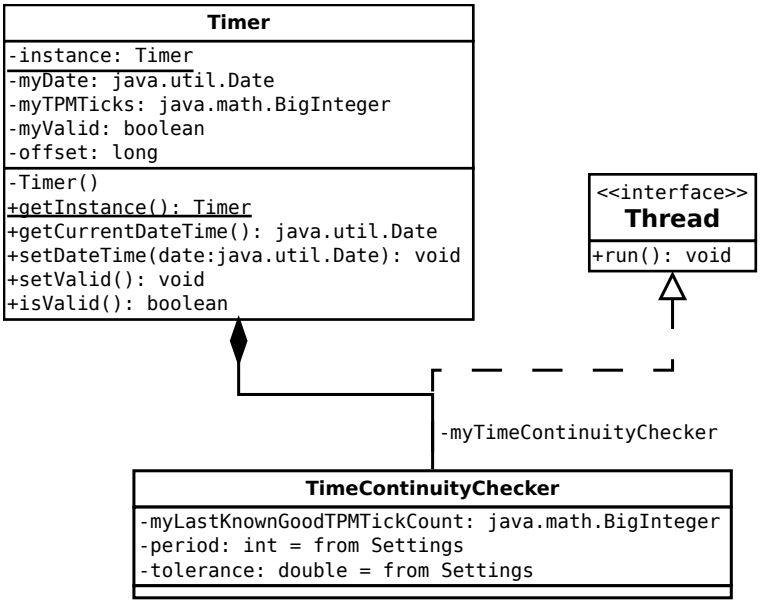


Figure 7.13.: Classes: Timer

7.6.5. Logger

The `Logger` collects statistics within the TvSM. Things like login-attempts, unsuccessful login-attempts, the number of executions of a certain cryptographic service and others are tracked. We implemented that by creating a singleton class called `Collector`. The `SampleProvider` interface allows the `CollectorThread` to collect frequent measurements. Prior to that, a `SampleProvider` has to register itself at the `Collector` class as active source. An illustration of the class layout is given in Figure 7.14. The collected statistics are written to disk and can be retrieved through the `MaintentanceSession`.

7.6.6. Summary

In this section we discussed the most important software blocks of the TvSM server implementation. That was the key slot solution, the scheduler and the task based architecture of the TvSM, the crypto engine, as well as the structures used to enable the presented time stamping service and the statistics collection.

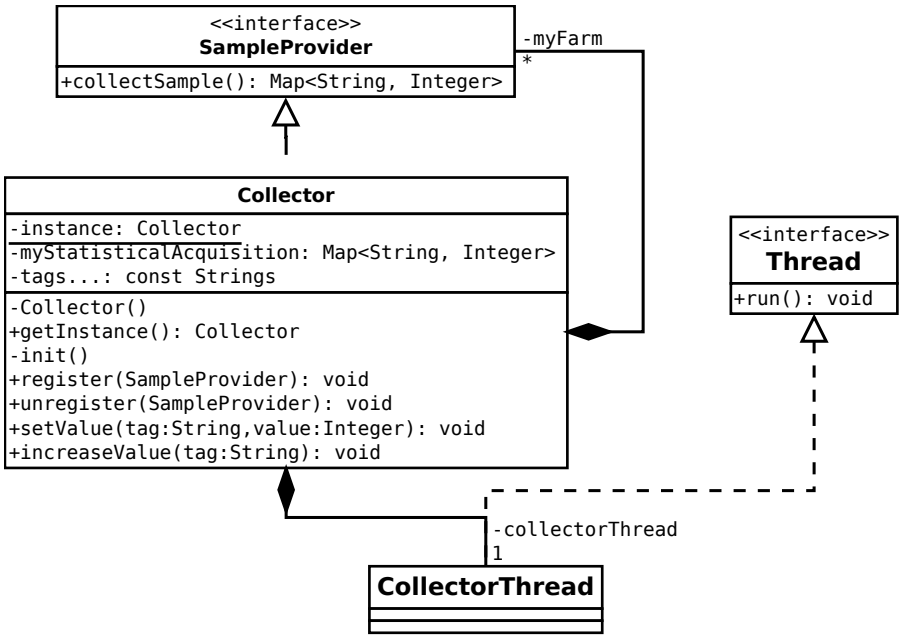


Figure 7.14.: Classes: Logger

8. Conclusion

The market analysis we did gave us an idea on what a security module is, what it does, how it does what it does, how fast and secure it does what is does, and on the impact of cost. The information gathered showed a split market. Commercial modules are either low-cost or high-cost and less secure or more secure respectively. There are few modules in between. The goal of this thesis therefore was to design a security module being low in cost but high in security. So, the result should add to the market perfectly. Technologies which are rarely used in security modules yet were necessary to achieve this. Besides the obvious area of cryptography, the technique of operating system virtualization as well as the relatively new trusted computing approach were used for our design and implementation.

We presented a Trusted virtual Security Module (TvSM) hardened by operating system virtualization, trusted computing, and a compact API, among others. The acTvSM platform [115] provides us secure boot, resource isolation and update management services. The narrow API gives little surface to attacks. The module's ID key is injected during module setup using a take ownership procedure. This key binds the TvSM instance to the host's TPM and can therefore be used for strong module identification. This important module key is a non-migratable TPM binding key and is therefore protected by the hardware component of trusted computing, namely the TPM. The master key is protected by the ID key and is therefore under the protection of the TPM as well. Moreover, the master key cannot be moved to another TvSM instance by a simple copy and paste.

The TvSM supports keys for software use only. Keys are protected by a parent key through key wrapping. Therefore, a key hierarchy is formed which is rooted by the master key and therefore by the ID key which is protected by the TPM. Each key has usage constraints as well as authentication data, namely the key secret. Knowledge of the key secret is necessary to use the key in cryptographic tasks.

Besides the standard key maintenance operations being creation, protection, destruction, we added key import/export, key migration, key backup, a time stamping service as well as an RSA signature creation engine to the feature set of the TvSM. Master key backup is done via inter-instance key migration. Key migration allows moving a key to another position in the key hierarchy. The target hierarchy can be local or within another TvSM instance. Our proof-of-concept implementation is capable of about 150 signature creation operations per second, which is slightly slower than the average security module of our market analysis. A more optimized implementation, be it preprocessing of time consuming computations or extensive use of real parallelism available on todays CPUs, for example, could boost the performance to a multiple of ten or more. Assuming a performance ten times higher, a TvSM would compete well with commercial security modules.

TvSM operation is organized as a role based system. There are two roles available: a privileged role for module maintenance and a standard role for module usage. Each role is protected by a corresponding role authentication secret. Authentication is done on a session

basis. The privileged role session offers master key management, time authority control as well as log and statistics retrieval. The standard role session allows a configurable amount of keys being loaded and used in cryptographic operations concurrently. A session timeout reduces vulnerability to denial of service attacks.

All in all, the presented TvSM is a step towards a new class of security modules, which provides a rather high level of security while being low at cost. Being a virtualized solution, the TvSM may fit existing infrastructures without the need for additional hardware. The good performance allows deployment up to medium size enterprises. Furthermore, operating the Common Criteria-approved TPM, a certification of the TvSM to an official standard may be possible as well.

A. Reviewed Security Modules

Product	FIPS-140 Level	Interface	Sources
AEP Networks K1200 HSM	4	Ethernet	[1, 106]
ARX Algorithmic Research PrivateServer	3	Ethernet	[3, 4]
Asigra Inc. AsigraEncModule Encryption Library	1	Library	[49]
Atos Worldline S.A./N.V. DEP/PCI v4	3	PCI	[50]
BeCrypt Ltd. Cryptographic Library v2	1	Library	[41]
Bloombase Info Security Company Cryptographic Module	1	Ethernet	[51]
Chunghwa Telecom Co., Ltd. HICOS PKI SmartCard	2	T=0, T=1, T=CL	[26]
CipherOptics ESG100 and ESG1002	2	Ethernet	[52]
Cisco PIX 525/535	1	Ethernet	[36]
Comtech Cryptographic Library v1.0	1	Library	[53]
Fortress Technologies FC-X	2	Ethernet	[56]
FutureX LLC ExCrypt CryptoModule	3	Ethernet, PCI	[37]
Gemalto Smart Guardian	3	USB	[57]
Gemplus GemXpresso Pro R3	3	T=0, T=1, T=CL	[6]
IBM eServer Crypto Coprocessor	4	PCI	[38, 106]
Imag Technologies Inc. TIMAC Module v1.0	3	UART	[28]
Oberthur Technologies ID-ONe Cosmo V7 SC Crypto Module	3	T=0, T=1, T=CL	[62]

Table A.1.: Reviewed security modules *continued on next page*

Product	FIPS-140 Level	Interface	Sources
Rainbow Technologies CryptoSwift HSM	3	PCI	[22]
SIC Crypto Toolkit		Library	[9]
SafeNet Inc. Luna CA4, CM2.4	3	Type 2 PC Card	[7]
SafeNet Inc. Luna PCI Crypo Module	2, 3	PCI	[33]
SafeNet Inc. Luna PCI7000	2,3	PCI	[8]
SafeNet Inc. ProtectServer Gold	3	PCI, UART	[59]
Spyrus Spyrus Hydra PC Locksmith	2	USB	[63]
Sun Microsystems Crypto Accelerator 6000	3	PCIe, UART, USB	[46, 10, 106]
Symantec Cryptographic Module v1.0	1	Library	[25]
Taua Biomatica Zyt Cryptographic Module	3	USB	[35]
Thales nCipher nShield miniHSM	3	UART, Parallel	[47]
Thales nCipher nToken	3	PCI	[48]
Thales nShield Connect	3	Ethernet	[11]
Utimaco Safeware AG CryptoServer 2000	3	Ethernet, PCI	[29]
bTrade LLC Security Module	1	Library	[61]
d'Crypt d'Cryptor ZE	3	UART	[30]

Table A.1.: Reviewed security modules

Product	asymmetric algorithms	symmetric algorithms	hash algorithms	HMAC options	RNG options
AEP Networks K1200 HSM	RSA, DSA, DH	AES, DES, TDES	MD5, SHA1, SHA2	-	FIPS 186-2
ARX Algorithmic Research PrivateServer	RSA	AES, DES, TDES	MD5, SHA1, SHA2	-	FIPS 186-2
Asigra Inc. AsigraEncModule Encryption Library	-	AES	SHA1, SHA2	SHA256	ANSI X9.31
Atos Worldline S.A./N.V. DEP/PCI v4	-	AES	SHA1, SHA2	-	yes
BeCrypt Ltd. Cryptographic Library v2	RSA	AES, DES, TDES, RC2	MD5, SHA1, SHA256	SHA256	ANSI X9.31
Bloombase Info Security Company Cryptographic Module	RSA	AES	SHA1, SHA2	SHA1, SHA2	ANSI X9.31
Chunghwa Telecom Co., Ltd. HICOS PKI SmartCard	RSA	AES, TDES	SHA1	SHA1	FIPS 186
CipherOptics ESG100 and ESG1002	RSA	AES	SHA1	SHA1	FIPS 186-2
Cisco PIX 525/535	DSA, DH	AES, DES, TDES, RC4	MD5, SHA1		ANSI X9.31
Comtech Cryptographic Library v1.0	-	AES, DES, TDES		SHA1	

Table A.2.: Reviewed security modules: cryptographic services *continued on next page*

Product	asymmetric algorithms	symmetric algorithms	hash algorithms	HMAC options	RNG options
Fortress Technologies FC-X	RSA, DH	AES, TDES	MD5, SHA1, SHA2	SHA1, SHA2	ANSI X9.31
FutureX LLC ExCrypt CryptoModule	RSA	DES, TDES	MD5, SHA1	SHA1	ANSI X9.31
Gemalto Smart Guardian	RSA	AES, TDES	SHA1, SHA256	SHA1	ANSI X9.31
Gemplus GemXpresso Pro R3	RSA	DES, TDES	SHA1	-	ANSI X9.31
IBM eServer Crypto Coprocessor	RSA, DSA	AES, DES, TDES	MD5, SHA1	-	FIPS 186-2
Imag Technologies Inc. TIMAC Module v1.0	-	AES	-	-	-
Oberthur Technologies ID-ONe Cosmo V7 SC Crypto Module	RSA, ECDSA	AES, TDES	SHA1, SHA2	-	ANSI X9.31
Rainbow Technologies CryptoSwift HSM	RSA, DSA	DES, TDES, RC4	MD5, SHA1	SHA1	ANSI X9.17
SIC Crypto Toolkit	various	various	various	various	various
SafeNet Inc. Luna CA4, CM2.4	RSA, DSA, ECDSA, DH	AES, DES, TDES, RC2, RC4, RC5, ARIA	MD2, MD5, SHA1, SHA2	MD5, SHA1	ANSI X9.17
SafeNet Inc. Luna PCI Crypo Module	RSA, DSA, ECDSA	AES, TDES	SHA1, SHA2	SHA1, SHA2	ANSI X9.31

Table A.2.: Reviewed security modules: cryptographic services *continued on next page*

Product	asymmetric algorithms	symmetric algorithms	hash algorithms	HMAC options	RNG options
SafeNet Inc. Luna PCI7000	RSA, DSA, ECDSA, DH	AES, DES, TDES, RC2, RC4, RC5, CAST3	MD2, MD5, SHA1, SHA2	MD5, SHA1	ANSI X9.31
SafeNet Inc. ProtectServer Gold	RSA, DSA, ECDSA, DH	AES, DES, TDES, RC2, RC4, SEED, ARIA	MD2, MD5, SHA1, SHA2, RIPEMD	IDEA, RC2, SEED, ARIA, MD2, MD5, SHA1, SHA2, RIPEMD	FIPS 186-2
Spyrus Spyrus Hydra PC Locksmith	ECDSA, ECDH	AES	SHA1, SHA2	-	FIPS 186-2
Sun Microsystems Crypto Accelerator 6000	RSA, DSA, ECDSA, DH	AES, TDES	SHA1, SHA512	SHA1, SHA512	FIPS 186-2
Symantec Cryptographic Module v1.0	-	AES, TDES	SHA1	SHA1	ANSI X9.31
Taua Biomatica Zyt Cryptographic Module	RSA	TDES	MD5, SHA1	-	ANSI X9.31
Thales nCipher nShield miniHSM	RSA, DSA, ECDSA, DH, ECDH, El-Gamal, KCDSA	AES, TDES, Blowfish, Arc Four, CAST5/6, Twofish, SEED	MD2, MD5, HAS-160, RIPEMD 160, SHA1, SHA2	MD2, MD5, RIPEMD-160, SHA1, SHA2	FIPS 186-2

Table A.2.: Reviewed security modules: cryptographic services *continued on next page*

Product	asymmetric algorithms	symmetric algorithms	hash algorithms	HMAC options	RNG options
Thales nCipher nToken	RSA, DSA, ECDSA, DH, ECDH, El-Gamal, KCDSA	AES, TDES, Blowfish, Arc Four, CAST5/6, Twofish, SEED	MD2, MD5, HAS-160, RIPEMD 160, SHA1, SHA2	MD2, MD5, RIPEMD-160, SHA1, SHA2	FIPS 186-2
Thales nShield Connect	RSA, DSA, ECDSA, DH, ECDH, El-Gamal, KCDSA	AES, DES, TDES, CAST, ARIA, Camellia, SEED	SHA1, SHA2	RIPEMD160	-
Utimaco Safeware AG CryptoServer 2000	RSA	DES, TDES, IDEA, Safer	MD5, MDC-2, RIPEMD-160, SHA1, SHA2	TDES	ANSI X9.31
bTrade LLC Security Module	RSA, DSA, DH	AES, TDES	MD5, SHA1, SHA2	SHA1, SHA2	ANSI X9.31
d'Crypt d'Cryptor ZE	-	AES, TDEA	SHA1	SHA1	ANSI X9.31

Table A.2.: Reviewed security modules: cryptographic services

Bibliography

[1] AEP Networks Series K Datasheet. http://www.aepnetworks.com/index.php/component/docman/doc_download/23-aep-series-k-k1200k300-product-datasheet (10 July 2010).

[2] AMD Virtualization Technology AMD-V. http://sites.amd.com/us/business/it-solutions/usage-models/virtualization/Pages/amd-v.aspx (02 April 2010).

[3] ARX PrivateServer HSM. http://www.arx.com/products/hsm (22 March 2010).

[4] ARX PrivateServer HSM Brochure. http://www.arx.com/files/Documents/PrivateServer-Brochure.pdf (23 March 2010).

[5] Austrian Citizen Card. http://www.buergerkarte.at/de/ueberblick/index.html (19 March 2010).

[6] GemPlus GemSafe Xpresso Description. http://www.petrotecard.pt/produtos/cartoes_sc_safex_en.htm (23 March 2010).

[7] SafeNet Inc. Luna CA4 Product Brief. http://www.safenet-inc.com/uploadedFiles/About_SafeNet/Resource_Library/Resource_Items/Product_Briefs_-_EDP/SafeNet_Product_Brief_Luna_CA4.pdf (24 March 2010).

[8] SafeNet Inc. Luna PCI-7000 Product Brief. http://www.safenet-inc.com/uploadedFiles/About_SafeNet/Resource_Library/Resource_Items/Product_Briefs_-_EDP/SafeNet_Product_Brief_Luna_PCI7000.pdf (24 March 2010).

[9] SIC Crypto Toolkit Website. http://jce.iaik.tugraz.at/ (24 March 2010).

[10] Sun Microsystems Crypto Accelerator 6000 Datasheet. http://www.oracle.com/ocom/groups/public/@ocompublic/documents/webcontent/036080.pdf (24 March 2010).

[11] Thales nShield Connect Datasheet. http://iss.thalesgroup.com/~/media/Files/Data%20Sheets/nShield%20Connect%20Datasheet.ashx (24 March 2010).

[12] Trusted Computing for the Java(tm) Platform. http://trustedjava.sourceforge.net/ (05 April 2010).

[13] Trusted Computing Group. http://www.trustedcomputinggroup.org/ (05 April 2010).

[14] VMware ESX 3.5. http://www.vmware.com/files/pdf/VMware-ESX-and-VMware-ESXi-DS-EN.pdf (02 April 2010).

[15] Website of bochs. http://bochs.sourceforge.net/ (02 April 2010).

[16] Website of DOSBox. http://www.dosbox.com/ (02 April 2010).

[17] Website of Linux VServer. http://linux-vserver.org/ (02 April 2010).

[18] Website of OpenVZ. http://wiki.openvz.org/ (02 April 2010).

[19] Website of QEMU. http://wiki.qemu.org (02 April 2010).

[20] Website of the Debian Linux Distribution. http://www.debian.org/ (31 October 2010).

[21] Security Requirements for Cryptographic Modules. FIPS 140-1, January 1994.

[22] Rainbow Technologies CryptoSwift HSM Security Policy. http://csrc.nist.gov/groups/STM/cmvp/documents/140-1/140sp/140sp162.pdf (24 March 2010), July 2001.

[23] Security Requirements for Cryptographic Modules. FIPS 140-2, May 2001.

[24] Intel® Low Pin Count (LPC) Interface Specification rev1.1. http://www.intel.com (12 November 2010), August 2002.

[25] Symantec Cryptographic Module v1.0 Security Policy. http://csrc.nist.gov/groups/STM/cmvp/documents/140-1/140sp/140sp483.pdf (24 March 2010), 2003,2004.

[26] Chunghwa Telecom Co., Ltd. HICOS PKI SmartCard Security Policy. http://csrc.nist.gov/groups/STM/cmvp/documents/140-1/140sp/140sp614.pdf (24 March 2010), December 2005.

[27] Evaluation criteria for IT security - Part 1: Introduction and general model. ISO/IEC 15408-1, 2005.

[28] Imag Technologies Inc. TIMAC Module v1.0 Security Policy. http://csrc.nist.gov/groups/STM/cmvp/documents/140-1/140sp/140sp507.pdf (24 March 2010), January 2005.

[29] Utimaco Safeware AG CryptoServer 2000 Security Policy. http://csrc.nist.gov/groups/STM/cmvp/documents/140-1/140sp/140sp543.pdf (24 March 2010), May 2005.

[30] d'Crypt d'Cryptor ZE Security Policy. http://csrc.nist.gov/groups/STM/cmvp/documents/140-1/140sp/140sp716.pdf (24 March 2010), October 2006.

[31] Identification Cards: Electrical interface and transmission protocols. ISO/IEC 7816-3, November 2006.

[32] Intel Trusted Execution Technology Architectural Overview. http://www.intel.com/technology/security/downloads/arch-overview.pdf (09 April 2010), September 2006.

[33] SafeNet Inc. Luna(R) PCI Cryptographic Module v2 Security Policy. http://csrc.nist.gov/groups/STM/cmvp/documents/140-1/140sp/140sp685.pdf (24 March 2010), June 2006.

[34] Security requirements for cryptographic modules. ISO/IEC 19790, 2006.

[35] Taua Biomatica Zyt Cryptographic Module Security Policy. http://csrc.nist.gov/groups/STM/cmvp/documents/140-1/140sp/140sp731.pdf (24 March 2010), December 2006.

[36] Cisco PIX 525/535 Security Policy. http://csrc.nist.gov/groups/STM/cmvp/documents/140-1/140sp/140sp758.pdf (24 March 2010), 2007.

[37] FutureX ExCrypt Cryptographic Module Security Policy. http://csrc.nist.gov/groups/STM/cmvp/documents/140-1/140sp/140sp764.pdf (24 March 2010), April 2007.

[38] IBM eServer Cryptographic coprocessor Security Module Security Policy. http://csrc.nist.gov/groups/STM/cmvp/documents/140-1/140sp/140sp661.pdf (24 March 2010), August 2007.

[39] TCG Specification: Architectural Overview rev1.4. http://www.trustedcomputinggroup.org (14.10.2010), August 2007.

[40] TPM Main Part 1 Design Principles 1.2 rev103. http://www.trustedcomputinggroup.org (14 October 2010), July 2007.

[41] BeCrypt Ltd. Cryptograpic Library v2 Security Policy. http://csrc.nist.gov/groups/STM/cmvp/documents/140-1/140sp/140sp1025.pdf (24 March 2010), September 2008.

[42] Evaluation criteria for IT security - Part 2: Security functional components. ISO/IEC 15408-2, August 2008.

[43] Evaluation criteria for IT security - Part 3: Security assurance components. ISO/IEC 15408-3, August 2008.

[44] Precision Clock Synchronization Protocol for Networked Measurement and Control Systems. IEEE 1599-2008, July 2008.

[45] Security requirements for cryptographic modules Cor 1. ISO/IEC 19790:2006/Cor 1, June 2008.

[46] Sun Microsystems Crypto Accelerator 6000 Security Policy. http://csrc.nist.gov/groups/STM/cmvp/documents/140-1/140sp/140sp1050.pdf (24 March 2010), 2008.

[47] Thales/nCipher nShield miniHSM Security Policy. http://csrc.nist.gov/groups/STM/cmvp/documents/140-1/140sp/140sp672.pdf (24 March 2010), June 2008.

[48] Thales/nCipher nToken Security Policy. http://csrc.nist.gov/groups/STM/cmvp/documents/140-1/140sp/140sp683.pdf (24 March 2010), June 2008.

[49] Asigra Inc. FIPS 140-2 Security Policy. http://csrc.nist.gov/groups/STM/cmvp/documents/140-1/140sp/140sp1240.pdf (24 March 2010), November 2009.

[50] Atos Worldline S.A./N.V. DEP/PCI v4 Security Policy. http://csrc.nist.gov/groups/STM/cmvp/documents/140-1/140sp/140sp1228.pdf (24 March 2010), November 2009.

[51] Bloombase Information Security Company Cryptographic Module Security Policy. http://csrc.nist.gov/groups/STM/cmvp/documents/140-1/140sp/140sp1241.pdf (24 March 2010), April 2009.

[52] CipherOptics ESG100 and ESG1002 Security Policy. http://csrc.nist.gov/groups/STM/cmvp/documents/140-1/140sp/140sp1250.pdf (24 March 2010), November 2009.

[53] Comtech Cryptographic Library v1.0 Security Policy. http://csrc.nist.gov/groups/STM/cmvp/documents/140-1/140sp/140sp1257.pdf (24 March 2010), July 2009.

[54] Digital Signature Standard (DSS). FIPS 186-3, June 2009.

[55] Evaluation criteria for IT security - Part 1: Introduction and general model. ISO/IEC 15408-1, 2009.

[56] Fortress Technologies FC-X Security Policy. http://csrc.nist.gov/groups/STM/cmvp/documents/140-1/140sp/140sp1251.pdf (24 March 2010), December 2009.

[57] Gemalto SmartGuardian Security Policy. http://csrc.nist.gov/groups/STM/cmvp/documents/140-1/140sp/140sp1260.pdf (24 March 2010), November 2009.

[58] IBM z/VM v6. http://www.vm.ibm.com/library/zvm610da.pdf (02 April 2010), October 2009.

[59] SafeNet Inc. ProtectServer Gold Security Policy. http://csrc.nist.gov/groups/STM/cmvp/documents/140-1/140sp/140sp739.pdf (24 March 2010), January 2009.

[60] Security Requirements for Cryptographic Modules (Revised Draft). FIPS 140-3 DRAFT, December 2009.

[61] bTrade, LLC Security Module Security Policy. http://csrc.nist.gov/groups/STM/cmvp/documents/140-1/140sp/140sp1261.pdf (24 March 2010), January 2010.

[62] Oberthur Technologies ID-One Cosmo V7-n Lite Security Policy. http://csrc.nist.gov/groups/STM/cmvp/documents/140-1/140sp/140sp1249.pdf (24 March 2010), July 2010.

[63] Spyrus Hydra PC Locksmith Security Policy. http://csrc.nist.gov/groups/STM/cmvp/documents/140-1/140sp/140sp1215.pdf (24 March 2010), March 2010.

[64] C. Adams. The CAST-128 Encryption Algorithm. RFC 2144, May 1997.

[65] C. Adams, P. Cain, D. Pinkas, and R. Zuccherato. Internet X.509 Public Key Infrastructure Time-Stamp Protocol (TSP). RFC 3161, August 2001.

[66] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield. Xen and the Art of Virtualization. In *SOSP '03: Proceedings of the nineteenth ACM symposium on operating system principles*, pages 164–177, ACM, New York, USA, 2003.

[67] A. Biryukov, C. D. Canniére, J. Lano, S. B. Ors, and B. Preneel. Security and Performance Analysis of ARIA. Technical report, Dept. Electrical Engineering-ESAT/SCD-COSIC, Katholieke Universiteit Leuven, Kasteelpark Arenberg 10, B-3001 Heverlee, Belgium, January 2004.

[68] G. R. Blakley. Safeguarding cryptographic keys. *International Workshop on Managing Requirements Knowledge*, 0:313, 1979.

[69] M. Bond. *Understanding Security APIs*. PhD thesis, University of Cambridge, 2004.

[70] M. Bond and R. J. Anderson. API-Level Attacks on Embedded Systems. *Computer*, 34:67–75, 2001.

[71] E. Brickell, J. Camenisch, and L. Chen. Direct anonymous attestation. In *Proceedings of the 11th ACM conference on Computer and Communications Security*, pages 132–145. ACM Press, 2004.

[72] D. Bruschi, L. Cavallaro, A. Lanzi, and M. Monga. Replay attack in TCG specification and solution. In *Computer Security Applications Conference, 21st Annual*, pages 11 pp. –137, December 2005.

[73] G. J. Chaitin. *Exploring Randomness*. Springer-Verlag New York, Inc., 2000. ISBN:1-85233-417-7.

[74] D. Challener, K. Yoder, F. Catherman, D. Safford, and L. V. Coorn. *A Practical Guide to Trusted Computing*. Pearson plc IBM press, 2008. ISBN-13: 978-0-13-239842-8.

[75] S. Chava. A Security Protocol for Multi-User Authentication. *The Computing Research Repository (CoRR)*, April 2008.

[76] L. Chen and M. Ryan. Attack, solution and verification for shared authorisation data in TCG TPM. In *Proceedings of the sixth International Workshop on Formal Aspects in Security and Trust (FAST2009)*, 2009.

[77] J. S. Clulow. The Design and Analysis of Cryptographic Application Programming Interfaces for Devices. Master's thesis, University of Natal, Durban, 2003.

[78] T. C. S. de Souza, J. E. Martina, and R. F. Costódio. Audit and backup procedures for Hardware Security Modules. In *IDtrust*, volume 283 of *Proceedings of the 7th symposium on Identity and trust on the Internet*, pages 89–97. ACM, 2008.

[79] T. Dierks and E. Rescorla. The Transport Layer Security (TLS) Protocol Version 1.2. IETF RFC 5246, August 2008.

[80] K. Dietrich and F. Röck. Performance Optimizations for DAA Signatures on Java enabled Platforms. *Journal of Universal Computer Science*, 16(4):519–529, 2009.

[81] S. Farrell. Why Didn't We Spot That? In *IEEE Internet Computing*, volume 14, pages 84–87. IEEE Educational Activities Department, January 2010.

[82] S. Fugkeaw, P. Manpanpanich, and S. Juntapremjitt. Exploiting X.509 Certificate and Multi-agent System Architecture for Role-Based Access Control and Authentication Management. In *7th IEEE International Conference on Computer and Information Technology*, pages 733 –738, 2007.

[83] D. Grawrock. *The Intel Safer Computing Initiative*. Building Blocks for Trusted Computing. Intel-Press, 2006. ISBN-13: 978-0976483267.

[84] J. Hoffstein, N. Howgrave-Graham, J. Pipher, J. Silverman, and W. Whyte. NTRUSign: Digital Signatures Using the NTRU Lattice. In M. Joye, editor, *Topics in Cryptology - CT-RSA 2003*, volume 2612 of *Lecture Notes in Computer Science*, pages 122–140. Springer Berlin / Heidelberg, NTRU Cryptosystems 5 Burlington Woods 02144 Burlington MA, 2003.

[85] IBM. VM History and Heritage. `http://www.vm.ibm.com/history/` (02 April 2010).

[86] T. M. Jones. Virtual Linux - An overview of virtualization methods, architectures, and implementations. In IBM developerWorks, December 2006.

[87] T. M. Jones. Discover the Linux Kernel Virtual Machine. In IBM developerWorks, April 2007.

[88] D. Kahn. *The Codebreakers*. Scribner, New York, revised edition, 1996.

[89] S. L. Kinney. *Trusted Platform Module Basics: using TPM in embedded systems*. Newnes, July 2006.

[90] A. Kivity, Y. Kamay, D. Laor, U. Lublin, and A. Liguori. KVM: the Linux Virtual Machine Monitor. In *OLS2007: Proceedings of the Linux Symposium*, pages 225–230, 2007.

[91] L. Lamport, R. Shostak, and M. Pease. The Byzantine Generals Problem. *ACM Trans. Program. Lang. Syst.*, 4:382–401, July 1982.

[92] H. Lee, S. Lee, J. Yoon, D. Cheon, and J. Lee. The SEED Encryption Algorithm. RFC 4269, December 2005.

[93] A. K. Lenstra and E. R. Verheul. An Overview of the XTR Public Key System. In *Public-Key Cryptography and Computational Number Theory*, page 2001. Verlages Walter de Gruyter, 2000.

[94] J. Lyle and A. Martin. Trusted Computing and Provenance: Better Together. In *Proceedings of the 2nd Workshop on the Theory and Practice of Provenance*, TAPP'10, pages 1–1, Berkeley, CA, USA, 2010. USENIX Association.

[95] E. H. McKinney. Generalized Birthday Problem. *The American Mathematical Monthly*, 73(4):385–387, April 1966.

[96] R. Mraz. Secure Blue: An Architecture for a Scalable, Reliable High Volume SSL Internet Server. Technical report, IBM Research Division, Thomas J. Watson Research Center, Yorktown Heights, NY 10598, June 2001.

[97] V. Mukhin. Multifactor authentication as a protection mechanism in computer networks. *Cybernetics and Systems Analysis*, 35:832–835, 1999.

[98] T. Müller. *Trusted Computing Systeme*. Springer-Verlag Berlin Heidelberg, 2008.

[99] G. Neiger, A. Santoni, F. Leung, D. Rodgers, and R. Uhlig. Intel(R) Virtualization Technology: Hardware Support for Efficient Processor Virtualization. *Intel Technology Journal*, 10(3):167–177, August 2006.

[100] M. Pirker and R. Toegl. Towards a virtual trusted platform. In *Journal of Universal Computer Science*, volume 16, pages 531–542. 2010.

[101] M. Pirker, R. Toegl, and M. Gissing. Dynamic Enforcement of Platform Integrity. In *Lecture Notes in Computer Science*, volume 6101 of *Trust and trustworthy Computing*, pages 265–272. 2010.

[102] B. Preneel. An Introduction to Cryptology. In B. Rovan, editor, *SOFSEM'98: Theory and Practice of Informatics*, volume 1521 of *Lecture Notes in Computer Science LNCS*, pages 204–221. Dept. Electrical Engineering-ESAT, Katholiek Universiteit Leuven, Belgium, 1998.

[103] E. Rescorla, M. Ray, S. Dispensa, and N. Oskov. The Transport Layer Security (TLS) Protocol Version 1.2. IETF RFC 5746, February 2010.

[104] A. R. Sadeghi and C. Stüble. Property-Based Attestation for Computing Platforms: Caring about Properties, not Mechanisms. In *New Security Paradigms Workshop*, 2004.

[105] K. Scarfone, M. Souppaya, and M. Sexton. *Guide to Storage Encryption Technologies for End User Devices*. National Institute of Standards and Technology, U.S. Department of Commerce, Computer Security Division, Information Technology laboratory, Gaithersburg, MD 20899-8930, November 2007.

[106] J. Schlyter. HSM - Hardware Security Modules. `http://www.iis.se/docs/hsm-20090529.pdf` (22 March 2010), May 2009.

[107] A. Shamir. How to share a secret. *Communications of the ACM*, 22:612–613, November 1979.

[108] X. Shiwei, Z. Huanguo, Y. Fei, X. Mingdi, and L. Zhide. Security Analysis of OIAP Implementation Based on BAN Logic. volume 1, pages 144 –148, November 2009.

[109] S. Spector. What is Xen Hypervisor. http://www.xen.org/files/Marketing/WhatisXen.pdf (09 April 2010), March 2010.

[110] G. Strongin. Trusted computing using AMD "Pacificy" and "Presidio" secure virtual machine technology. In *Information Security Tech. Report*, volume 10, pages 120–132. Elsevier Advanced Technology Publications, Oxford, UK, January 2005.

[111] Sun. JCE Standard Names. http://java.sun.com/j2se/1.4.2/docs/guide/security/CryptoSpec.html#AppA (26 April 2010), 1996-2002.

[112] T. Takebe. Trend in Security Evaluation and Accreditation. In *SICE Annual conference 2008*, pages 1482–1486, Tokyo, Japan, Oktober 2008. Industrial Automation Systems Business, Yokogawa Electric Corporation.

[113] K. T. F. Team. The Korean Certifcate-based Digital Signature Algorithm, August 1998.

[114] R. Toegl, M. Pirker, and G. Fliess. acTvSM Deliverable 4.1: Draft API Design. Unpublished, 2010.

[115] R. Toegl, M. Pirker, and M. Gissing. acTvSM: A Dynamic Virtualization Platform for Enforcement of Application Integrity. In *Proceedings of INTRUST 2010*. Springer Verlag, 2010. accepted for publication.

[116] S. Tomasco and C. Zinkowski. IBM Extends Enhanced Data Security to Consumer Electronics Products. http://www-03.ibm.com/press/us/en/pressrelease/19527.wss (09 April 2010), April 2006.

[117] G. S. Vernam. *Cipher Printing Telegraph Systems*. June 1926.

[118] VMware. Understanding Full Virtualization, Paravirtualization and Hardware Assist. http://www.vmware.com/files/pdf/VMware_paravirtualization.pdf (02 April 2010), September 2007.

[119] T. Winkler and B. Rinner. Applications of trusted computing in pervasive smart camera networks. In *WESS '09: Proceedings of the 4th Workshop on Embedded Systems Security*, pages 1–10, New York, NY, USA, 2009. ACM.

[120] A. C. Yao. Theory and Applications of Trapdoor Functions. In *IEEE Symposium on Foundations of Computer Science*, volume 23, pages 80–91. IEEE, 1982.

[121] P. Youn. The Analysis of Cryptographic APIs Using Formal Methods. Master's thesis, Massachusetts Institute of Technology, 2004.

[122] P. Youn, B. Adida, M. Bond, J. Clulow, J. Herzog, A. Lin, R. L. Rivest, and R. Anderson. Robbing the bank with a theorem prover. Technical Report UCAM-CL-TR-644, University of Cambridge, Computer Laboratory, August 2005.