Project acronym: OVERSEE

Project title: Open Vehicular Secure Platform

Project ID: 248333

Call ID: FP7-ICT-2009-4

Programme: 7th Framework Programme for Research and Technological Development

Objective: ICT-2009.6.1: ICT for Safety and Energy Efficiency in Mobility

Contract type: Collaborative project

Duration: 01-01-2010 to 30-06-2012 (30 months)

**Deliverable** **D3.3:**

**Security Services**

**Architecture and Implementation Description**

Authors: Hakan Cankaya (escrypt)

Thomas Enderle (escrypt)

(Fraunhofer)

(TUB)

Reviewers:

Dissemination level: Public

Deliverable type: Report

Version: 0.3

Submission date: 30th of February 2012

# Abstract

This document provides a detailed description of the OVERSEE platform security services. It describes the individual modules, the overall architecture and the interfaces to these services. The document contains specification of the individual modules, in case of standard interfaces external documents are referenced, and more detailed need of specification will be done in dedicated external documents.

*Note:*

*With the new project plan of OVERSEE this document is due to the end of February 2012. This version is incomplete and unreviewed, please consider this document as a general feedback only*.

# Contents

[Abstract ii](#_Toc316962788)

[Contents iii](#_Toc316962789)

[List of Tables v](#_Toc316962790)

[List of Figures vi](#_Toc316962791)

[List of Abbreviations vii](#_Toc316962792)

[Document History viii](#_Toc316962793)

[1 Introduction 1](#_Toc316962794)

[1.1 Scope and Objective of the Document 1](#_Toc316962795)

[1.2 Rationale and Approach 1](#_Toc316962796)

[2 Security Architecture 3](#_Toc316962797)

[2.1 OVERSEE Security Service Architecture 3](#_Toc316962798)

[3 Specification and Configuration of the Security Components 7](#_Toc316962799)

[3.1 EVITA HSM 7](#_Toc316962800)

[3.2 Sharing Cryptographic Resources of the HSM 7](#_Toc316962801)

[3.2.1 EVITA PKCS#11 Wrapper 7](#_Toc316962802)

[3.2.2 PKCS#11 Client 7](#_Toc316962803)

[3.2.3 PKCS#11 Proxy 8](#_Toc316962804)

[3.3 Central Authentication and Authorization Servers 8](#_Toc316962805)

[3.3.1 LDAP 8](#_Toc316962806)

[3.3.2 Sign-On Services 9](#_Toc316962807)

[3.3.3 Authentication services for runtime environments without TCP/IP support 9](#_Toc316962808)

[3.4 Central Secure Policy Insertion 9](#_Toc316962809)

[3.5 Secure SW Management 9](#_Toc316962810)

[3.6 Secure Storage 9](#_Toc316962811)

[3.7 ITS Components 9](#_Toc316962812)

[3.7.1 Filtering/ Subscription Model 10](#_Toc316962813)

[3.7.2 Event/Message Types 11](#_Toc316962814)

[3.7.3 Vehicle State 11](#_Toc316962815)

[3.8 Secure Boot 12](#_Toc316962816)

[3.8.1 Secure boot vs. authenticated boot 12](#_Toc316962817)

[3.8.2 Platform Configuration Registers 12](#_Toc316962818)

[3.8.3 Measurement 12](#_Toc316962819)

[3.8.4 Application to OVERSEE 13](#_Toc316962820)

[3.8.5 The Oversee Boot Process 13](#_Toc316962821)

[3.8.6 Securing the OVERSEE Boot Process 13](#_Toc316962822)

[3.8.7 Updating PCR Values 15](#_Toc316962823)

[3.9 Attestation services 15](#_Toc316962824)

[3.9.1 Attestation to external entities 15](#_Toc316962825)

[3.9.2 Attestation to Applications Running on OVERSEE 15](#_Toc316962826)

[4 Security Services 16](#_Toc316962827)

[5 Conclusion 17](#_Toc316962828)

[6 References 18](#_Toc316962829)

# List of Tables

[Table 1: Policy Insertion Ticket Structure 9](#_Toc316962169)

# List of Figures

[Figure 1 - OVERSEE Security Architecture 3](#_Toc316962216)

[Figure 2 ITS-FL structure 10](#_Toc316962217)

[Figure 3 Filter Example 10](#_Toc316962218)

[Figure 4 ITS-FL subscriptions 11](#_Toc316962219)

# List of Abbreviations

CAM Co-operative Awareness Message

CU Communication Unit

ECU Electronic Control Unit

EV Emergency vehicle

GWN Global Wireless Networks

HMI Human Machine Interface

ITS Intelligent Transport Systems

IVN In-Vehicle Network

LWN Local wireless network

OEM Original Equipment Manufacturer

PKI Public Key Infrastructure

PoC Proof of Concept

PS Positioning service

RE Runtime environment

RSU Road side unit

SM Security Module

SVAS Secure Vehicle Access Service

UN User networks

V2V Vehicle-to-vehicle

V2X Vehicle-to-vehicle or Vehicle-to-Infrastructure

# Document History

|  |  |  |
| --- | --- | --- |
| **Version** | **Date** | **Changes** |
| 0.3 | 13-02-2012 | Draft Version |

# Introduction

## Scope and Objective of the Document

The objective of this deliverable is to describe the implementation of the security services architecture of the OVERSEE platform.

This deliverable is the result of Task 3.3 and reflects the implementation of the prototype platform but also provides detailed specification of solutions which may be not implemented in the prototype. These modules will be indicated in the document.

*WP2 results serve as the main design input for this specification, nevertheless the solutions introduced here are not meant to realize a complete implementation of the OVERSEE design.*

*Thus, the task 3.3 comprises the realization of the*

1. *Integration of a hardware security module (HSM) and a secure mechanism to access the services of the HSM in the multi-partitioned architecture of OVERSEE.*
2. *Realization of higher level security services which can be accessed by applications.*
3. *Proof of Concept realization of an ITS communication stack*
4. *Various security related services like secure sw update and secure storage.*

This deliverable is a draft version and does not involve all the security modules yet, furthermore the specification for many modules is not detailed yet.

The D3.3 will reflect implementation changes until the end of the project and can therefore be revised in a later stage.

*Note:*

*With the new project plan of OVERSEE this document is due to the end of February 2012. This version is incomplete and unreviewed, please consider this document as a general feedback only*.

## Rationale and Approach

The document follows an up-to-down approach and starts with a general overview of the security architecture and approach in OVERSEE. Afterwards the individual components are specified. The individual components are either specified in this document or a detailed specification of the component is referenced. Furthermore the version of the component and the dependencies to other components are listed. The general specification of the components follows a configuration specific to the OVERSEE architecture. Finally the individual interfaces provided to the users of the OVERSEE Platform are specified.

The document is structured as follows:

* Section 1 Introduction
* Section 2 Security Architecture
* Section 3 Specification and Configuration of the Security Components
* Section 4 Security Services

# Security Architecture

This section provides a general overview of the security services and the underlying architecture of OVERSEE.

## OVERSEE Security Service Architecture

The virtualized architecture of OVERSEE enables different runtime environments to run parallel on the same platform with different levels of trustworthiness. Access to the various resources of the system is restricted by the OVERSEE architecture. This enables the creation of secure and isolated services which can be reached over dedicated channels. The need for integrity and trustworthiness can be limited to a minimal number of modules in this way and evaluated separately from the user specific part of the OVERSEE platform. Based on such a trusted base further enhanced security services can be built upon. In this part the general architecture of the OVERSEE security concept will be explained and the possibilities to assure system integrity with this architecture will be discussed. Furthermore some of the individual services which can be built upon this security concept will be summarized.

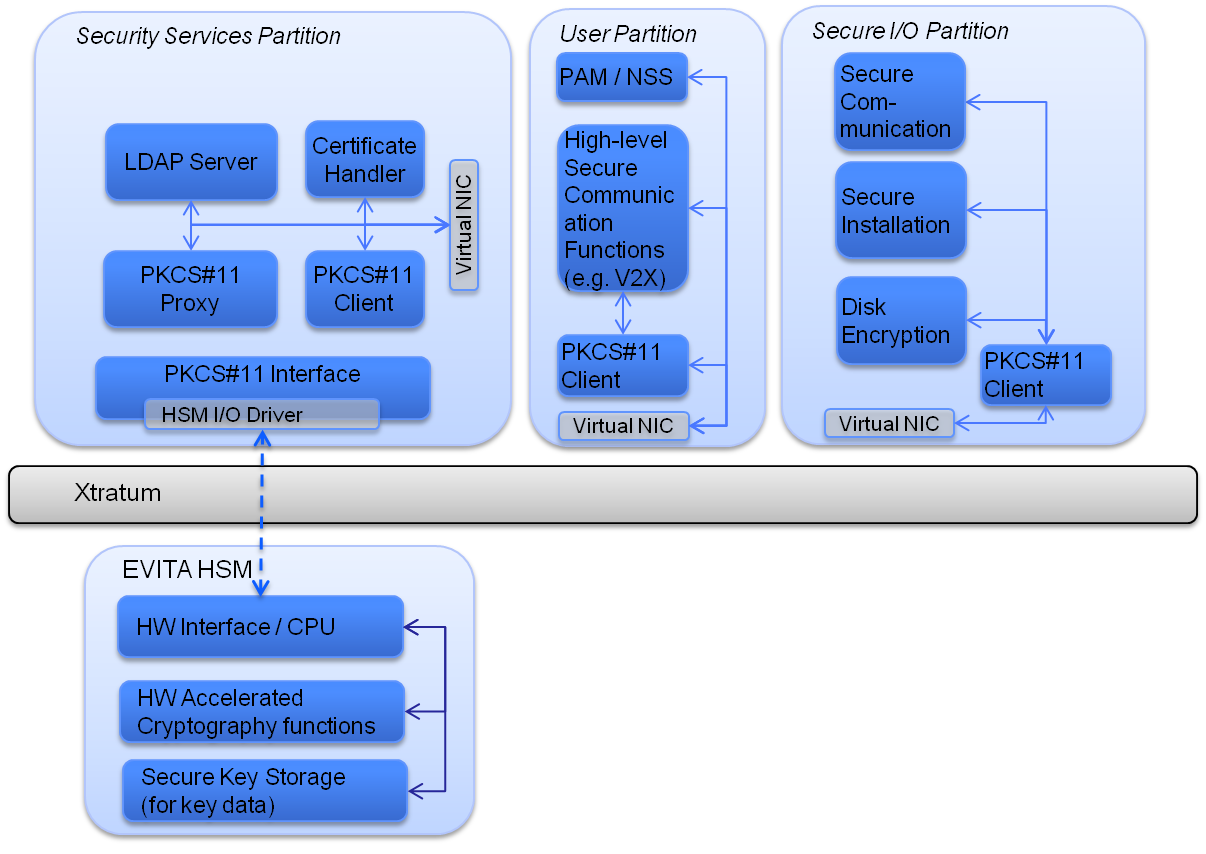


Figure 1 - OVERSEE Security Architecture

As seen in Figure 1 the OVERSEE architecture involves a hardware security module. In the first version of the OVERSEE architecture the implementation is based on the EVITA HSM (Hardware Security Module) (7), which provides many essential services and features serving as a base for the security concept of OVERSEE. The HSM is logically coupled to the security service partition of the OVERSEE platform which provides a secure and isolated runtime environment for security services in general that can be requested by the other partitions through secured communication channels.

As mentioned before a trusted configuration of the system is essential to build the security concept upon. To assure this a secure boot process assisted with the HSM is integrated in the OVERSEE architecture. The EVITA HSM adheres to a large extend to the TPM specification (9) and enhances this specification with further features, creating an essential part of the secure boot functionality.

The secure boot process starts with the authentication of the XtratuM hypervisor and the two essential partitions, the secure I/O partition and the security services partition. The hash values of the actual configuration of these partititons of the OVERSEE architecture are compared with the known configuration values of the platform in the HSM. Based on this comparison many actions can be taken and/or enforced by the architecture. One action would be to restrict access to cryptographic keys by coupling the correct hash value of the system configuration to the access right of cryptographic keys stored in the HSM. Furthermore the HSM can provide signed attestations of the system configuration which can be used by external entities to remotely validate the integrity of the system. The EVITA HSM provides multiple such configuration hash registers which may be used to detail the integrity validation of the system. This feature can be used to validate only the integrity of one specific runtime environment (or only a part of it) with each register and take actions only for this specific part of the OVERSEE platform.

The parallel execution of the runtime environments in an integrated architecture enables also other approaches. A known issue with today’s systems is that cold start of IT systems is not very common anymore; the system starts usually from a stand-by or hibernated state which excludes the whole secure boot process. Furthermore the fact that harmful changes can only be recognized in the next boot process already can cause severe security flaws. The parallel execution of the security service partition can react on such situation more dynamically. The integrity of specific memory areas or stored data can be validated cyclic on-the-run or after specific actions like warm-start or software installation. The actions upon recognition of non-expected configuration can vary from just notifying related modules to stopping the tampered partition from running.

The security of stored data in the system is assured by storage encryption and individual services for encrypting individual files. A possible candidate for encrypting file systems would be dm-crypt. The key material for the encryption is stored in the HSM, providing a sealed storage. The file system of the user partitions are provided by the secure I/O partition using Virtio (6). Thus enabling a seamless integration by forwarding centrally encrypted/decrypted file systems to the user partitions.

OVERSEE architecture provides a central point for handling software installations. This central instance can be used for a variety of use-cases and business models providing a mandatory verification procedure in means of authorization, integrity, authenticity, compatibility and dependencies. The software package structure is based on a standardized format, in our case the Debian package (10). The central handler serves as a proxy between the partitions and the repositories and validates the packages before forwarding to the destination and initiating an installation. It also provides functionality for updating whole partition images or the hypervisor.

The direct services provided by the security service partition can be summarized as follow:

* A controlled interface to cryptographic functions and key material hosted by the HSM.
* Central handling of authentication and authorization information.
* Certificate handling.

OVERSEE aims to base on standardized interfaces and provide a modular design. Therefore the interface to the security module is based on the PKCS#11 (11) specification which is supported by most of the security modules (eg. smart cards, hardware security modules etc.). This enables to also use other security hardware instead of the EVITA HSM with minimal effort. The PKCS#11 interface is tunneled to the other partitions by a proxy communicating with a PKCS#11 client driver hosted at the other partitions. The OVERSEE PKCS11#-proxy design enables parallel access to the cryptographic functions and objects of the security hardware. It also provides a layer for restricting access of the individual services for each partition sending a request over the PKCS11#-proxy. The layer can filter the requests for a various number of parameters as the requested cryptographic object, requesting partition, authentication or authorization status of a specific user. This architecture enables a very flexible and modular design to fit the system to the needs of the platform designer.

The central handling of authentication and authorization data is essential for the security and flexibility of the system. To enable such a service an LDAP (Lightweight Directory Access Protocol) server (e.g. openLDAP ) is provided by the security service partition. This server can be invoked by the other partitions by NSS (Name Switch Service) or LDAP based PAMs (Pluggable Authentication Modules). Also direct usage of the LDAP server through look-up services can be used to retrieve data to validate information as authorization or roles of a specific user, partition or any other entity. Further functionalities like single sign-on are built upon this infrastructure. The access right to the LDAP server and individual data is restricted in the partition level.

The security service partition also provides services for handling certificates and security tokens like certificate validation and creation. Furthermore services for importing security attributes or objects (e.g. cryptographic keys, new users) are provided by the certificate handler. The secure key storage of the HSM enables the storage of trusted public keys (e.g. OEM public key) at the beginning of the vehicle lifecycle, enabling a chain of trust.

Most of the concepts listed here are adaptations of existing and standardized solutions to the OVERSEE architecture. Consequently many of the approaches are based on Linux systems and IP communication. To avoid to block the integration of non-Unix systems into the OVERSEE platform a general approach is taken in the OVERSEE design. The XtratuM hypervisor provides secure channels (queueing channels, shared memory etc) between partitions which can be configured individually. A non-conform communication standard can always be built upon these channels interfacing with the services in the security partition. For example this would be a solution to provide the cryptographic services to partitions not supporting a PKCS#11-driver. The PKCS#11-client driver would be hosted by the security service partition in such a case and listen to the dedicated XtratuM channel for the specific requests from the non-Linux partition. An elegant solution for OSEK OS partitions would be the mapping of the Virtual Function Bus to these security services. To sum up, the connection to the security services is always a point to point connection, usually consisting of sequential communication packages and no need for complicated package tracking or error resilience algorithms and therefore can be realized with little effort in many ways using the communication facilities of XtratuM as a basis..

# Specification and Configuration of the Security Components

## EVITA HSM

TBD. Specification of related EVITA API.

## Sharing Cryptographic Resources of the HSM

For the prototype the consortium decided to integrate the EVITA HSM as the cryptographic hardware module. The EVITA HSM provides a detailed API for interfacing the features. Still, to provide a generic solution and ease the migration to another security module in the future we decided to share the security module with a more standardized interface than the native EVITA API, the PKCS#11 API. This API is used in many smart cards and also large scale hardware security modules and is a de facto standard in this area.

### EVITA PKCS#11 Wrapper

To provide PKCS#11 capabilities to the EVITA HSM a wrapper is built around the EVITA driver. The wrapper is based on the open source project opencryptoki driver framework (14). The softHSM configuration of the opencryptoki module is used as a base. This configuration uses openSSL as the core library for executing the cryptographic functions. We took this configuration and exchanged the openSSL functions with the EVITA HSM access functions. Another issue was the object (e.g. key) management of the opencryptoki module. This was solved by saving the attributes of the keys in the opencryptoki framework but the data itself in the HSM. For the proof-of-concept implementation the following functionality is supported:

* key generation
* key import
* EC signing
* EC verifying

### PKCS#11 Client

We also want to use the PKCS#11 interface as the standard interface for the user partitions to access cryptographic services. Therefore a PKCS#11 interface is provided for the user partitions. In our prototype implementation we implemented a PKCS#11 driver for Linux which provides a standard API to the user runtime environment. For every application using the PKCS#11 interface, a separate instance of the client library is loaded. This instance provides a standard API PKCS#11 interface to the application. Underneath, the library instance creates a TCP connection to the PKCS#11 Proxy on the security service partition. For every library call, it serializes all arguments, and transmits it to the proxy.

The proxy sends the request to the HSM, receives the answer and forwards it to the client. This process is transparent to the application. Also, for every thread of an application, a separate library instance and proxy connection is created..

### PKCS#11 Proxy

The PKCS#11 Proxy runs on the security partition. It listens on a TCP socket for connections from client libraries. Upon an incoming connection, the following steps are taken:

* First the source is determined in order to be able to separate requests from different partitions.
* Then the application identifier is read from the connection, in order to maintain an intra-partition subdivision
* If the connection belongs to a new application, a child process is created, which loads the underlying PKCS#11 library. If the connection belongs to an application, which is already running, the connection is forwarded to the existing child process.
* The child process first deserializes the incoming PKCS#11 request including the parameters.
* Depending on the kind of request, and the arguments of the request, policy decisions are made, and masking is applied.
  + For every operation that uses or reads an existing key object, the child process verifies, if the partition has the permission to do that specific kind of operation on that object. The permission information is taken from the object’s PKCS#11 label.
  + For every operation that creates or modifies to a key object, resource restrictions are applied and it is ensured that the key object is labelled with the permission information.
* The proxy calls the according function of the underlying binary PKCS#11 library. Then the following operations are carried out for the result:
  + For operations that list key objects, all objects are filtered according to their permissions.
  + TBD
* The filtered result is then serialized and transmitted back to the requesting partition via the existing TCP connection.

## Central Authentication and Authorization Servers

### LDAP

TBD

TBD

### Sign-On Services

TBD

### Authentication services for runtime environments without TCP/IP support

TBD

## Central Secure Policy Insertion

OVERSEE provides a central point to insert security policies which can be enforced by the dedicated modules. Basically the central policy insertion point accepts dedicated tickets which are signed by an authority (e.g. OEM). The signature of the ticket is validated and the content of the ticket is imported into the directory service. The new policy or authorization can be notified to the related module if the dedicated field in the ticket is set.

The tickets provide the following fields.

|  |  |
| --- | --- |
| Ticket issuer | The authority creating the ticket (e.g. OEM) |
| Validity of ticket | Date until which the ticket is valid. |
| Policy | A policy. In our PoC an LDAP entry |
| Notification list | List of Addresses to be notified |
| Key Information | Key ID of public key |
| Signature | Generated signature |

Table : Policy Insertion Ticket Structure

## Secure SW Management

TBD

## Secure Storage

TBD

## ITS Components

This section presents the concepts and design of the ITS facility layer (**ITS-FL**). ITS-FL aggregates information from SVAS and mobile router to offer a unified perspective to V2X applications. ITS-FL is a java library and employs remote method invocation (RMI) to interconnect to V2X applications on other partitions.

Figure 2 depicts the structure of the ITS-FL. ITS-FL is connected to the mobile router to receive incoming messages and send out messages (i.e., CAM, DENM and application specific messages). For instance, ITS-FL periodically sends CAMs describing the own vehicle state that is provided by SVAS. Furthermore, V2X Applications can subscribe to incoming CAMs, DENMs or application specific messages and send out DENMs. In addition, V2x applications are able to query the actual vehicle state that has been gathered by the ITS-FL.



Figure ITS-FL structure

ITS-FL offers a flexible and lightweight subscription mechanism to receive messages coming from the mobile router as described by the next section.

### Filtering/ Subscription Model

ITS-FL offers a flexible configurable subscription mechanism of incoming V2X messages to minimize the amount of messages needed to be processed by interested V2X applications. Hereby, applications can simply specify required network message types. These message subscriptions are based on filters that can be combined into complex filters via Boolean filter aggregators.



Figure 3 Filter Example

Figure 3 shows such a complex filter that is composed of an “AND filter” that aggregates a “DENM filter” and a “distance filter”. The DENM filter accepts all incoming DENMs but no other message types. The “distance filter” accepts all messages that where originated by stations that are within defined vicinity around the receiving station. Both filters are combined into a complex filter via the “AND filter” that accepts messages for which all aggregated filters apply. There are several filters that can be combined to achieve desired behaviour:

AND, OR, NOT, CAM, DENM, DENM Cause code, Distance, Relevance area, Heading, Sender node id.

On the one hand, this filter subscription approach enables V2X applications to conveniently specify message types of their interest. On the other hand, it helps to minimize the amount of data transferred over the RMI protocol from one partition to another.



Figure 4 ITS-FL subscriptions

Figure 4 shows a V2X application that subscribes itself to incoming messages by specifying a filter and a call-back that will be invoked whenever a new message arrives that matches the given filter. Thereafter, the mobile router receives a new message that matches the given filter and ITS-FL notifies the previously subscribed V2X application about this new message.

### Event/Message Types

There are three types of events: *DENMs*, *CAMs* and *application specific*. CAMs and DENMs can be filtered by the before mentioned filter subscription mechanism. Application specific messages are handled by an own subscription mechanism that is comparable but simpler. Application specific messages can be filtered by their numeric type.

### Vehicle State

ITS-FL offers a unified perspective to the state of the vehicle. On the one hand, this aggregated state is employed to periodically broadcast CAMs via mobile router. On the other hand, V2X applications can observe the aggregated state or change it. E.g., an application that manages a light bar on top of the own vehicle can modify its state when it has changed.

Here is a short excerpt of the available vehicle state properties:

LIGHT\_BAR\_IN\_USE, VEHICLE\_SPEED, LONG\_ACCELERATION, YAW\_RATE, HEADING, DIRECTION\_OF\_DRIVING, ENGINE\_SPEED, ALTITUDE, GEO\_POSITION, FUEL\_CONSUMPTION

## Secure Boot

### Secure boot vs. authenticated boot

The boot process generally consists of multiple stages. Every stage loads the next stage into memory and executes it. For secure boot every stage has to ensure, that the next stage in the chain is authentic, before it transfers control to it. If not, it has to take some action, for example create log entries, or halt the system.

In contrast, for authenticated boot, the boot process will continue even if the verification fails, but certain secret data or certain services will not be available for the system for the whole session.

The chain must have a physically secured trust anchor at the beginning; otherwise, an attacker could simply manipulate the software of the first stage and manipulate all further checks. In the classical x86 environment this can be fulfilled by a TPM BIOS.

### Platform Configuration Registers

A Platform Configuration Register (PCR) is a hash value stored in a protected place (e.g. an HSM or a TPM), together with a reference value. Commonly there are several PCR implemented, some of them for special purposes and hard wired to some internal functionality, others for user-defined usage.

The only possible way to modify a given PCR from the outside is to “extend” it with a Hash Value (or “Measurement”) Mn+1, which is defined as follows

Extend(PCR(n), Mn+1) : PCR(n+1) = HASH(PCR(n) ||Mn+1)

Due to the limited operation, and the fact that the hash function is a one-way function, the following property holds true: Given a certain PCR start value and an arbitrary chosen target value, it is computationally infeasible to calculate a measurement or a series of measurements, so that the extension of the start value with these measurements would result in that chosen target state of the PCR.

This property guarantees with a sufficiently high probability: If the value of a PCR is equal to the reference value, all the measurements that were extended into the PCR must have been the same as the reference measurements used to calculate the reference value.

### Measurement

Stage Sn loads the next stage Sn+1 from disk, measures it, and executes it. The measurement process is conducted as follows:

First stage Sn calculates a hash Mn+1 of stage Sn+1. Then stage Sn extends a PCR by this hash value.

This is done for every stage in the boot process. Secret keys or special operations are only made available, if the PCR value is equal to a certain reference value.

### Application to OVERSEE

The integrity of OVERSEE encompasses the following parts:

* Integrity of the oversee platform itself
  + Integrity of the XtratuM kernel
  + Integrity of oversee service partition(s)
* Integrity of guest partitions

The former has to be enforced by the oversee platform. In order to enforce the latter, oversee will provide all the necessary services.

### The Oversee Boot Process

Following we identify the stages of the OVERSEE boot process:

1. The BIOS or firmware is started
2. The BIOS or firmware loads a boot loader from disk and executes it.
3. The boot loader loads the XtratuM binary from disk and executes it.
4. XtratuM first runs with a boot plan and starts only the necessary service partitions
5. XtratuM switches to the main plan and starts all other partitions

In order to implement authenticated boot, every such stage has to be verified by the previous stage.

### Securing the OVERSEE Boot Process

In the following, we describe the secure boot process for each stage.

#### Stage 1: BIOS/Firmware

In order to guarantee real security, the BIOS or firmware itself must be secured by hardware. In a final implementation is done by a secure hardware element right from the start. In a final embedded system this would be preferably be the HSM which is directly embedded into the processor.

As our demonstration secure hardware element (EVITA-HSM) is connected via ethernet, such a service can only be provided at a stage, where ethernet and HSM drivers are available. The earliest possible time for access is after the linux kernel and initrd of the service partition are loaded.

In order to secure a demonstration system until the point in time when the HSM is available, in an x86 environment the services of a TPM could be used as a fallback solution. Unfortunately, our demonstration hardware does not provide a TPM. Nevertheless we will prepare the use of a TPM, in order to show principal feasibility.

#### Stage 2: Bootloader

As oversee already uses the GRUB boot loader, we will use a modification called “Trusted GRUB”.

Trusted GRUB is basically an older version of GRUB that has been enhanced by the following functions:

* Measurements of its composing parts
* Measurement of the binary boot image, here the XtratuM binary, including all partition kernels.
* Measurement of all boot parameters.
* Measurement of a small number of predefined files.

Therefore, it is also possible to measure XtratuM configuration files as well as initrd files and small read-only file systems of the service partition.

Again, as there is no TPM available on our demonstration hardware, these measurements will be done, but not finally verified.

#### Stage 3: XtratuM binary

The bootloader measures the composed XtratuM binary and any dependencies (e.g. configuration files).

#### Stage 4: Security service / IO partition

The Security Service Partition consists of a linux kernel, an initrd and a filesystem. The kernel is directly included in the XtratuM binary and therefore already measured by the bootloader. Measurements of the initrd must also be done by the bootloader.

If the measurements are successful, the HSM (or TPM) unlocks usage of secret keys. In case of a combined TPM/HSM approach, one of those keys can be used to access the HSM.

For the main partition and user data, we will use an encrypted file system using such a secret key. This grants confidentiality and limited integrity protection (i.e. protection against directed manipulation of file system blocks, not against destruction of blocks or against restoring of previously saved blocks).

#### Stage 5: Other partitions

As soon as the security partition is running, it supports the measurement of the remaining partitions. This will be independent of PCR/ECR registers.

Several Possibilities are offered:

* (implicit) Verification of bare partition
* Verification of complete provided block device.

This is only possible for very small block devices.

* Verification of Initrd.
* Verification of boot parameters
* Providing Secure Storage for limited integrity.

TBD

On success, XtratuM switches to the main plan.

### Updating PCR Values

When system updates are applied, previous PCR reference values will not fit anymore. Therefore, also PCR reference values have to be updated.

There are two possibilities:

In a secure environment, the system can be fully booted up into the desired state without protection. The resulting PCR value simply can be taken as the reference value.

When performing an update in an insecure environment, an appropriate new value has to be calculated before boot and then written into the PCR reference.

As the PCR calculation process is exactly specified, it is in feasible to pre-calculate a new reference value. This value can be supplied by the update or calculated on the fly by the system during an update.

## Attestation services

TBD

### Attestation to external entities

### Attestation to Applications Running on OVERSEE

# Security Services

TBD

# Conclusion

# References

x

|  |  |
| --- | --- |
|  | 1. Commission of the European communities, "Action Plan for the Deployment of Intelligent Transport Systems in Europe", December 2008 2. ETSI, "European Profile Standard for the Physical and Medium Access Control Layer of Intelligent Transport Systems Operating in the 5Ghz Frequency Band", ES 202 663 V110 3. Open Vehicular Secure Platform (OVERSEE) Project, http://www.oversee-project.com 4. XtratuM Hypervisor designed for real-time embedded systems, http://www.xtratum.org 5. Whitaker A. and Cox R.S. and Shaw M. and Gribble S.D, "Rethinking the Design of Virtual Machine Monitors", *Computer vol.38 no.5*, IEEE, May 2005, pp. 57-62 6. Russell R., "virtio: Towards a De-Facto Standard For Virtual I/O Devices", *ACM SIGOPS Operating Systems Review - Research and developments in the Linux kernel* 7. E-safety vehicle intrusion protected applications (EVITA) Project, http://www.evita-project.org 8. *Volume 42 Issue 5*, ACM, July 2008Weyl Benyamin, Wolf Marko et.al., “Deliverable D3.2: Secure On-board Architecture Specification”, 21. December 2010, 9. “TPM Main Specification Level 2 Version 1.2, Revision 116”. Available at http://www.trustedcomputinggroup.org/resources/tpm\_main\_specification 10. “Basics of the Debian Package Management System”. Available at http://www.debian.org/doc/FAQ/ch-pkg\_basics.en.html 11. RSA Laboratories, “PKCS #11 v2.20: Cryptographic Token Interface Standard”, 28 June 2004 12. “OpenLDAP”, Available at http://www.openldap.org/ 13. Schaub F. and Ma Z. and Kargel F., "Privacy Requirements in Vehicular Communication Systems*", International Conference on Computational Science and Engineering (Vancouver, 2009)*, IEEE 14. http://sourceforge.net/projects/opencryptoki/ |

x