oversee

SEVENTH FRAMEWORK PROGRAMME

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 D2.2:

Specification of security services incl. virtualization and firewall mechanisms

Authors:	Jan Holle (Uni Siegen) André Groll (Uni Siegen) Alfons Crespo (UPVLC) Hakan Cankaya (escrypt) Thomas Enderle (escrypt) Nicholas Mc Guire (OpenTech) Andreas Platschek (OpenTech)
Reviewers:	Hakan Cankaya (escrypt) Thomas Enderle (escrypt) Florian Friederici (FOKUS)

Dissemination level:	Public
Deliverable type:	Report
Version:	1.6
Submission date:	11 November 2013

Abstract

This document specifies all capabilities and services of OVERSEE, which are related to security. Thus, it defines a large part of the OVERSEE design since the project has a strong focus on security.

Security services that are part of the generic OVERSEE implementation will be depicted. These services are offered to the partitions through an appropriate API and are reused by other OVERSEE components, too. Additionally, the following security related features are specified: Whole virtualization concept, OVERSEE firewall, secure assurance of prioritized resource access and internal communication requirements in the virtualized environment.

Contents

List of Figures

List of Tables

List of Acronyms and Abbreviations

2G	Second generation mobile phone system
3G	Third generation mobile phone system
ABM	Asymmetric Bi-directional mapping
AES	Advanced Encryption Standard
AMT	Abstract Machine Test
API	Application Programming Interface
ARINC	Aeronautical Radio Incorporated
BIOS	Basic Input Output System
CA	Certification Authority
CAM	Cooperative Awareness Message
CAN	Controller–area Network
CBC	Cipher Block Chaining
CBC-MAC	Cipher Block Chaining Message Authentication Code
CC	Common Criteria
CCM	Counter with CBC-MAC
CEN	European Committee for Standardization
CMAC	Cipher based MAC
CPU	Central Processing Unit
CRL	Certification revocation list
DENM	Decentralized Environmental Notification Message
DoS	Denial of Service
DSRC	Dedicated Short-Range Communications
DTD	Document Type Definition
eCall	Emergency Call
ECB	Electronic Code Book
ECC	Elliptic Curve Cryptography
ECR	ECU configuration registers
ECU	Electronic Control Unit
FiFo	First in First out
FPU	Floating point unit
GCM	Galois/Counter Mode

GNU	GNU's not Unix
GPS	Global Positioning System
HAL	Hardware Abstraction Layer
HMAC	Hash based MAC
HMI	Human Machine Interface
HSM	Hardware Security Module
HW	Hardware
I/O	Input Output
ICT	Information and Communication Technology
IMA	Modular Avionics
IP	Internet Protocol
IPC	Inter Process Communication
ITS	Intelligent Transportation System
LDM	Local Dynamic Map
MAC	Message Authentication Code
MILS	Multiple Independent Levels of Security and Safety
MMU	Memory Management Unit
NIC	Network Interface Card
NIST	National Institute of Standards and Technology
OEM	Original Equipment Manufacturer
OS	Operating System
OSEK	Offene Systeme und deren Schnittstellen für die Elektronik im Kraftfahrzeug
PIFP	Partitioned Information Flow Policy
PKCS	Public Key Cryptography Standards
PKI	Public Key Infrastructure
PoS	Positioning Service
POSIX	Portable Operating System Interface (for Unix)
PRNG	Pseudo Random Number Generator
PSAP	Public-Safety Answering Point
QoS	Quality of Service
RAM	Random-access memory
RTE	Runtime Environemnt
RTEMS	Real-Time Operating System for Multiprocessor Systems
RTOS	Real-time Operating System

SecS	Security Services
SKPP	Separation Kernel Protection Profile
SSL	Secure Sockets Layer
SSPM	Security Services Provider Module
SVAS	Secure Vehicle Access Service
TLS	Transport Layer Security
TOE	Target Of Evaluation
TSF	TOE Security Functions
USB	Universal Serial Bus
V2X	Vehicle-to-X
VDX	Vehicle Distributed eXecutive
VFB	Virtual Function Bus
VM	Virtual Machine/Virtual Machine Manager
VMA	Virtual Memory Area
VMM	Virtual Machine Manager/Monitor
Wi-Fi	Wireless Fidelity
XEX	Xor-Encrypt-Xor
XML	Extensible Markup Language
XPTE	Exchange Page Table Entry
XTS	XEX-based Tweaked CodeBook

1 Introduction

The Open Vehicular Secure Platform (OVERSEE) project has produced this deliverable; therefore it contains contributions from all partners, while the main contributors are: escrypt, UPVLC, OpenTech and University of Siegen.

The present document contains the specification of the security relevant features and capabilities of OVERSEE, hence a large part of the overall specification, since OVERSEE has a dedicated focus on security for open platforms.

1.1 Scope and Objectives of the Document

The scope of this document is the specification of the security related features and capabilities of OVERSSE. Since OVERSEE is a project focusing on security from secure communication aspects to secure parallel execution of applications, this document contains a large part of the overall OVERSEE design.

The objective of the actual document is to provide a global description for the implementation of the security related parts of OVERSEE. Thus, the document would be used as a kind of functional specification for the implementation phase and also a reference book concerning the security capabilities of the platform during usage.

1.2 Definitions

For the purpose of the present document, the terms and definitions given in [1], [2] and [3] apply, if not otherwise noted.

1.3 Document Outline

The document is structured as follows: Section 1 gives an introduction on the scope and intentions. Section 2 specifies the security services provided by OVERSEE. Section 3 is concerned with the description of the overall virtualization concept. Section 4 contains the specification of the OVERSEE firewall concept. Section 5 is dedicated to the secure assurance of prioritized access on shared resources and finally section 6 specifies the internal communication requirements in the virtualised environment.

2 Security Services

2.1 Overall Structure of Security Services

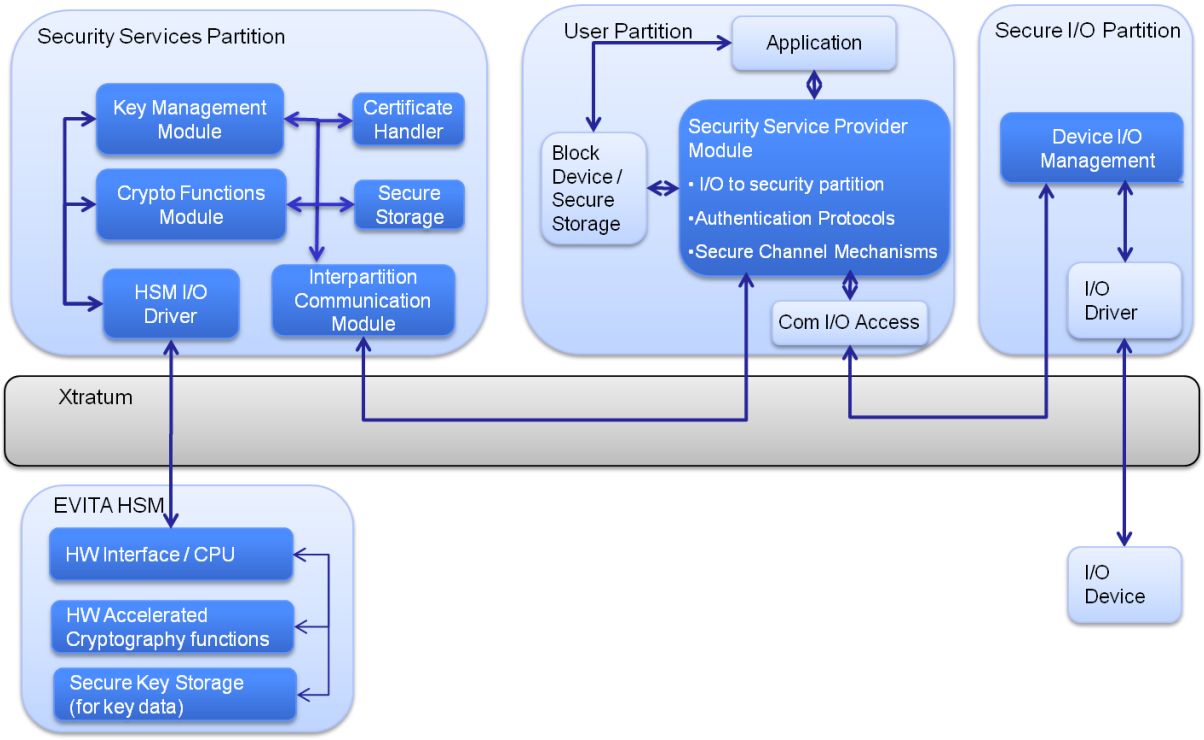

Figure 1: Structure of Security Services

Two main facts play a key role defining the requirements of the security services. First of all OVERSEE aims to provide a platform for ICT applications, which require a set of dependable security services. The second aspect is that OVERSEE provides multiple isolated runtime environments. This demands a strong level of isolation, which also has an impact on the security services.

The OVERSEE security architecture basically consists of three parts. The first part is the hardware security module (HSM). The HSM provides accelerated cryptographic function execution, secure key and certificate storage, and registers for secure boot services. The second part is a dedicated partition for the security services. This partition owns exclusive rights to access the HSM and also hosts building blocks for further security services. The last part is the so-called security service provider module (SSPM). This module is a high level client located at the user partitions (or any other partition if needed) providing the link between user applications and security services.

2.2 Hardware Security Module

The HSM is not part of the OVERSEE design however a crucial element in the security architecture. This part of the document will still list some requirements for the HSM on which the security services of OVERSEE will be build on.

Furthermore it is also possible to implement most of the functionality of a HSM in the security partition as a software version, though respectively in a less secure way. The design of OVERSEE is based on the existence of a HSM but the structure of the security partition will be specified in a flexible way to enable also a software implementation of HSM functionalities.

2.2.1 HSM Cryptographic Functions

The cryptographic functions provided by the HSM are hardware-accelerated implementations of the algorithm thus offering higher performance than software implementations. In addition the HSM enables a secure environment for the cryptographic functions making any tampering attempts practically impossible. Furthermore key data used for the algorithms are also handled internally in the HSM, thus assuring security of key data.

In order to satisfy the requirements to sign, verify and encrypt messages for V2X communications and PKI Infrastructure communications the following minimum set of functions are proposed by the OVERSEE design to be supported by the HSM:

- ECC-256: Asymmetric cryptographic engine based on elliptic curve arithmetic.
- Whirlpool: A hash function proposed by NIST.
- AES-128: A symmetric block encryption standard with a key length of 128 bits. It should support standard block encryption modes as ECB and CBC but also advanced modes as GCM and CCM.
- PRNG: A pseudo random number generator.
- Counters: Counter management with authorization.

An implementation of further needed cryptographic functions into the security services partition would be a solution for creating a flexible and expandable design but is not a very secure method regarding key security. Therefore the HSM should also provide enhancement capabilities regarding cryptographic functions. That is the firmware of the HSM should enable realization of further cryptographic functions, even if supporting hardware blocks do not exist in the HSM. This would enable the implementation of cryptographic functions in software, e.g., ECC-224 as a candidate for cryptography in the upcoming ITS standards for Europe.

2.2.2 HSM Secure Key Storage

The HSM features an internal storage for keys and certificates. The internal storage is not accessible directly over any interface, hence creating a secure storage for security relevant data. The exportation of a key from the HSM depends on the configuration of the relevant key. For further information on the security of key material and key management refer to Deliverable 2.4 [4].

2.2.3 HSM Secure Boot Registers

The HSM features so called ECU configuration registers (ECR) and assigned functions on which a secure and/or authenticated boot process can be built on. The following functionalities are provided in this context. Further information on secure boot process can be found in chapter 2.5.

- **Extend ECR:** This function is used for updating the ECR with a new (hash) value. The new value is pro-vided as input and chained together with the existing value stored in the ECR using a hash update function.
- **Retrieve ECR:** For any ECR, this function may be used to retrieve and verify its actual value. The ECR value is signed by a certification key whose usage authorization data needs to be presented as input. Furthermore, a nonce value, possibly obtained as a challenge from an external verifier, may be included to be able to prove the freshness of the signature. The ECR value together with its signature are returned as output
- **Preset ECR:** This function is used to manage references to ECR values by ECR indices in the context of secure boot. The function provides the possibilities to directly update an ECR with a new value, to set a new reference to an ECR index ("inherit") or to remove an existing reference ("clear").
- **Compare ECR:** This function allows the direct comparison of the current ECR with a reference. If match fails, an automatic secure boot alarm may be activated. As a precondition, a reference must be set for the given ECR index, and an internal secure boot failure mechanism must be given.

2.2.4 HSM Hardware Interface

The HW interface provided by the HSM is the dedicated interface to reach the functionality of the HSM. The interface depends on the HSM and the OVERSEE platform and will not be specified any further in the OVERSEE design. The proof of concept implementation will use any interface provided by the HSM without concerning any security aspects although the HW interface is an important communication path in the design respective security. Later implementations of the platform have to assure a secure interface; for further information refer to the document D2.4 [4].

2.3 Security Services Partition

The security services partition provides a secure and isolated runtime for the shared security services. Multiple partitions thus multiple applications will access the security functions of the platform. The security partition provides a trustable anchor in the whole security architecture and a controlled access to the security services as user partitions (and applications) have only restricted access to the security partition and services provided by the partition.

In this section of the document the building blocks of the security partition are listed and described. Further specification of the building blocks is left to the implementation phase of the platform. The API to the security services are listed in section 2.4 where the client module for user partitions is specified.

2.3.1 Security Partition Binding

The security services partition binding module is the first instance interfacing requests from the other partitions. The connection between the other partition and the security partition binding module exists of a queue in each direction for the commands and a shared memory for exchanging large chunks of data.

The command structure received by the binding module can be seen in Table 1. The request ID is a unique label defined by the other partition for internal use at the other partition. This ID is followed by the command id and the parameters of the command. The security partition binding module handles the incoming command by adding the origin of the request and a unique handling id to the command before forwarding the command to the appropriate module.

No.	Name	Description
1	Request ID	A unique ID given by the user partition for the request.
2	Command ID	ID of requested command.
3	Command parameters	Parameters needed for the command. Details are specified in every method.

Table 1: Security Service General Command Structure

2.3.2 Hardware Security Module I/O Driver

The HSM I/O driver is the interface to the HSM module. The driver is provided by the HSM manufacturer and therefore is out of scope of the OVERSEE design. The driver should provide a seamless integration of the HSM services into the runtime environment and function as a non-blocking device.

2.3.3 Cryptographic Services Module

Any request for a cryptographic function is handled by the cryptographic functions module. The request is forwarded from the security services partition binding module and handled basically in this sequence:

- Check for authorization of requesting partition for requested service.
- Check for authorization of requested key and key usage.

Subsequently the cryptographic service on the HSM is triggered or a software implementation of the requested cryptographic function is executed with the needed parameters.

2.3.4 Key Management Module

The Key Management Module plays an important role in the security architecture of the OVERSEE design. Issues as authenticity, integrity and encryption are closely coupled to ownership of key information as the isolation of the security services for the multiple

partitions is mainly provided by defining different access rights to the key material. A key request is strictly handled by the key management module. The requesting partition has to be authorized for the requested key and the requested usage of the key. Only in this case the key data may be used for the requested security service by the user partition. The reliability of the origin of the request is assumed to be provided by the virtualization layer.

Information regarding key management can be found in D2.4 Specification of Secure Communication [4].

An important role of the key management module is to integrate the key management functionality of the HSM with the virtualized environment of OVERSEE. Basically the key management of the HSM is not aware of the multiple runtime environments. The key management module builds an extra management stage between the user partition and the HSM providing the needed access control.

The key management module also handles key export, import and creation requests from the other partitions. The API for the functionality is specified in chapter 2.4 Security Services Provider Module. Table 2: Internal Key Structure shows the internal key structure used in OVERSEE, Table 3 shows the encrypted key blob structure which is used in case of key importation/exportation.

No	Name	Description
1	Algorithm Identifier	This parameter determines the algorithm to be used with the key
2	Use Flags	This parameter indicates the operations that may be performed with the key.
3	Key ID	Identifier of the key.
4	Validity	Indicates the last valid date the key can be used.
5	Certification Data	This parameter is optional and may be used to certify the origin and/or properties of the key (e.g., by MVK). It may consist of, but is not limited to a digital signature.
6	Authorization Data	This vector represents the authorization data items for each individual. (Passwords, list of authorized partitions)
7	Public Key Data	Public key data bytes.
8	Private Key Data	Private key data bytes.
9	Key handle	Internal handle.

Table 2: Internal Key Structure

No	Name	Description
1	Algorithm Identifier	This parameter determines the algorithm to be used with the key
2	Use Flags	This parameter indicates the operations that may be performed with the key.
3	Key ID	Identifier of the key.
4	Validity	Indicates the last valid date the key can be used.
5	Public Key Data	Public key data bytes.
6	Encrypted Blob	This part of the key structure includes all confidential key material encrypted with corresponding transport key.
7	Authentication Code	This parameter is used to protect the integrity and authenticity of the key data. It may be either a MAC created by the symmetric transport key or a digital signature w/ IDK.

Table 3: Key Structure for Export/Import

2.3.5 Secure Storage

The secure storage module is responsible for encrypting and decrypting files. The encryption procedure is described in the following steps.

1. The file to be encrypted arrives at the module from the user partition.
2. A random key K_R is generated.
3. K_R is used to encrypt the file.
4. K_R is encrypted with the requested storage key K_S. The user partition has to have the usage rights of this key.
5. The encrypted file, the encrypted key K_R and the applied encryption parameters are bundled together as a file and sent back to the user partition.

The API for the functionality is specified in chapter 2.4.3.

2.3.6 Certificate Handler

The certificate handler is responsible for creating new certificates and handling incoming certificates. Certificates can be stored by this module. Additionally, the access control management for certificates is also done within the certificate handler module.

2.3.7 Security Services Configuration Table

As mentioned before access to security services and key data requires proper access rights of the partitions. The security service configuration table provides a configuration tool to manage the access rights of the partitions.

Accessing and changing the table have to be done in a secure way to avoid any unauthorized changes in the table. The user has to prove authorization by a challenge-response process. Furthermore the table should also be not accessible offline. The table is either stored in the HSM or is encrypted before storing. The content of table should also be signed to avoid any offline changes in the table.

2.4 Security Services Provider Module

The security services provider module (SSPM) is located at the user partition and is the client for accessing the security services provided by the security services partition. Furthermore, the SSPM provides high-level functionality for several services like communication protocols and secure storage. This means that only when services provided by the security services partition have to be accessed the client contacts the security services partition and requests the required service.

The SSPM is an application level service running in the user partition. The connection to the security services partition is provided by the virtualization layer and consists of a queue for the requests and a shared memory area for exchanging larger chunks of data. The general command structure between the SSPM and security partition is described in section Security Partition Binding of this document. The next sections describe different services and the associated API's provided by the SSPM.

2.4.1 Cryptographic Services API

Following are the methods provided by the cryptographic services module.

2.4.1.1 Cipher

The cipher API provides the required method for general encryption/decryption. The algorithm and operation mode can be defined with the parameters.

No	Name	Description
1	Algorithm identifier	Reference to associated symmetric algorithm
2	Cipher mode	Indicate decryption or encryption mode
3	operation mode	Indicate cipher mode of operation {ECB|CBC|GCM|EAX|..}
4	padding_scheme	Indicate padding scheme {none|bit|byte|pkcsx|..}
5	IV_SIZE	Size of given initialization vector (can be 0)
6	Key handle	Handle (internal ID) of the requested key
7	Key Authorization Size	Size of authorization value of the key
8	Key Authorization Value	Authorization value of key
9	Input Data	Pointer to input data
10	Input Data Size	Size of input data

Table 4: Cipher API

No	Name	Description
1	Output Data	Pointer to Output data
2	Output Data Size	Size of Output data

Table 5: Cipher API, Output

2.4.1.2 CMAC

The CMAC API provides the required method for cipher based MAC calculation. The algorithm and operation mode can be defined with the parameters.

No	Name	Description
1	Algorithm identifier	Reference to associated symmetric algorithm
2	MAC mode	Indicate MAC mode: sign, timestamped sign or verify
3	operation mode	Indicate cipher mode of operation {CMAC/OMAC/…}
4	padding_scheme	Indicate padding scheme {none|bit|byte|pkcsx|..}
5	Key handle	Handle (internal ID) of the requested key
6	Key Authorization Size	Size of authorization value of the key
7	Key Authorization Value	Authorization value of key
8	Input Data	Pointer to input data
9	Input Data Size	Size of input data

Table 6: CMAC API

No	Name	Description
1	Output Data	Pointer to Output data
2	Output Data Size	Size of Output data

Table 7: CMAC API, Output

2.4.1.3 Hash and HMAC

The HMAC API provides the required method for general hash calculation and hash based MAC calculation. The algorithm and operation mode can be defined with the parameters.

No	Name	Description
1	Requesting Partition	ID of partition requesting the service
2	Algorithm identifier	Reference to associated symmetric algorithm
3	Hash mode	Indicate hash or HMAC creation mode
4	Key handle	Handle (internal ID) of the requested key
5	Key Authorization Size	Size of authorization value of the key
6	Key Authorization Value	Authorization value of key
7	Input Data	Pointer to input data (to be verified MAC, or Data for creating MAC)
8	Input Data Size	Size of input data

Table 8: Hash and HMAC API

No	Name	Description
1	Output Data	Pointer to Output data
2	Output Data Size	Size of Output data

Table 9: Hash and HMAC API, Output

2.4.1.4 Signature Generation

The following method is used to handle the generation of signatures with a private key. This method is applied to the complete message to be signed.

No	Name	Description
1	Requesting Partition	ID of partition requesting the service
2	Algorithm identifier	Reference to associated asymmetric algorithm
3	Hash algorithm identifier	Reference to associated hash algorithm
4	padding_scheme	Indicate padding scheme {none|bit|byte|pkcsx|..}
5	Time stamp	Indicate additional timestamp creation
6	Key handle	Handle (internal ID) of the requested key
7	Key Authorization Size	Size of authorization value of the key
8	Key Authorization Value	Authorization value of key
9	Input Data	Pointer to input data (to be verified MAC, or Data for creating MAC)
10	Input Data Size	Size of input data

Table 10: Signature Generation API

No	Name	Description
1	Output Data	Pointer to Output data
2	Output Data Size	Size of Output data

Table 11: Signature Generation API, Output

2.4.1.5 Signature Verification

The following method is used for signature verification. These functions are applied to the complete message whose signature shall be verified.

No	Name	Description
1	Algorithm identifier	Reference to associated asymmetric algorithm
2	Hash algorithm identifier	Reference to associated hash algorithm
3	padding_scheme	Indicate padding scheme {none|bit|byte|pkcsx|..}
4	Time stamp	Indicate additional timestamp creation
5	Key handle	Handle (internal ID) of the requested key
6	Key Authorization Size	Size of authorization value of the key
7	Key Authorization Value	Authorization value of key
8	Input Data	Pointer to input data (to be verified MAC, or Data for creating MAC)
9	Input Data Size	Size of input data

Table 12: Signature Verification API

No	Name	Description
1	Output Data	Pointer to Output data
2	Output Data Size	Size of Output data

Table 13: Signature Verification API, Output

2.4.1.6 Random Number Generation

This method provides random number generation functionality.

No	Name	Description
1	Algorithm identifier	Reference to associated asymmetric algorithm
2	Requested Random bytes	Size of requested random array

Table 14: Random Number Generator API

No	Name	Description
1	Output Data	Pointer to Output data
2	Output Data Size	Size of Output data

Table 15: Random Number Generator API, Output

2.4.1.7 Counters

The counter methods provide API's for creating, reading incrementing and deleting counters.

No	Name	Description
1	access_authorization_value	counter access authorization value (i.e., password hash)

Table 16: Counter API, Create counter

No	Name	Description
1	access_authorization_value	counter access authorization value (i.e., password hash)

Table 17: Counter API, Delete counter

No	Name	Description
1	Counter ID	counter id of counter to be incremented

Table 18: Counter API, Output value create counter

No	Name	Description
1	counter_identifier	counter id of counter to be incremented
2	access_authorization_value	counter access authorization value (i.e., password hash)
3	counter_incrementation	counter incrementation value

Table 19: Counter API, Increment counter

No	Name	Description
1	counter_identifier	counter id of counter

Table 20: Counter API, Read counter

No	Name	Description
1	Current Value	counter id of counter to be incremented

Table 21: Counter API, Output value read counter

2.4.2 Key Management API

The key management API provides the necessary methods for key creation, importation and exportation. Furthermore this API provides services for creating and handling incoming certificates.

2.4.3 Secure Storage API

With the secure storage API files can be stored in a secure and encrypted manner in the user partition. The process of encryption is explained in chapter 2.3.5. The secure storage API mainly serves as an interface.

No	Name	Description
1	Requesting Partition	ID of partition requesting the service
2	Data/File ID	reference to associated asymmetric algorithm
3	Create Key	Indicate new key creation for encryption
4	Key handle	Handle (internal ID) of the requested key for the random key encryption
5	Key Authorization Size	Size of authorization value of the key
6	Key Authorization Value	Authorization value of key
7	Input Data/File	Pointer to input data
8	Input Data Size	Size of input data

Table 22: File encryption API

No	Name	Description
1	Output Data/File	Pointer to Output data
2	Output Data Size	Size of Output data

Table 23: File encryption API, Output

No	Name	Description
1	Requesting Partition	ID of partition requesting the service
2	Data/File ID	reference to associated asymmetric algorithm
3	Key handle	Handle (internal ID) of the requested key
4	Key Authorization Size	Size of authorization value of the key
5	Key Authorization Value	Authorization value of key
6	Output Data/File	Pointer to Output data
7	Output Data Size	Size of Output data

Table 24: File decryption API

No	Name	Description
1	Output Data/File	Pointer to Output data
2	Output Data Size	Size of Output data

Table 25: File decryption API, Output

2.4.4 Alternative Secure Storage

Additionally to the standard API a pseudo file system driver provides the user with a more convenient way to use the secure storage facility. The file system then acts as a transparent encryption layer for the user. Files written into the file system are encrypted in the background and written into a designated folder. Vice versa any read attempt also triggers a decryption of the requested file.

This has some disadvantages: it is not appropriate for large files or random access on parts of files.

Therefore, we will propose another alternative storage facility: One introduces a transparent encryption layer on ordinary block devices. This works as follows:

The platform allocates a fixed amount of non-volatile storage to a block device. This is then encrypted using a common encryption algorithm for storage encryption, i.e. AES in XTS mode [21] and made available to a user partition as a pseudo device. That means every low level block the user partition writes to the pseudo device is transparently encrypted and then written to the real block device, whereas any block read on the pseudo device results in a read in the real device following a decryption.

At this time neither such a secure file system nor a secure block device will explicitly be specified by oversee, but because of their transparency, we expect that it can easily be retrofitted on existing applications.

2.4.5 Secure Communication API

The secure communication functionality of the SSPM aims to provide high level protocols for security related communication. The module will interface with the virtual network interface in the user partition and act as an application level protocol layer. Each protocol will listen to dedicated port number. The protocols will be executed as a part of the SSPM in the user partition and access the security partition functionality when needed.

The secure communication functionality of the SSPM also provides a flexible framework for the developer to integrate their own protocols into the building block, though OVERSEE will also provide some standard protocols within the design.

- Diffie-Hellman Key Agreement PKCS #3
- Certification Request Standard PKCS #10
- TLS / SSL

2.5 Secure Boot

The trustworthiness and security of the services provided by OVERSEE depend highly on the authenticity and integrity of the platform. As defined by Peter Neumann “An object is trustworthy if and only if it is proven to operate as expected”. There are several existing procedures to assure a trusted platform like secure boot, authentic boot or remote attestation.

The architecture of the OVERSEE platform enables various kinds of secure boot procedures. The selection and integration of a secure boot process is a critical design decision depending on many aspects like timing requirements, deployment of software, chosen runtime environments, chosen hardware platform and security requirements. Therefore we will not specify a specific and detailed secure boot process in this section but have a look on different techniques and options suitable for the OVERSEE architecture. The section aims to serve as a guide for the developer.

2.5.1 Guideline for Providing Secure Boot in OVERSEE

There are many methods to realize a secure boot process and the options depend also very much on the hardware platform used to implement OVERSEE. Still we will provide a guideline to realize a secure boot mechanism into OVERSEE.

The key point of a secure boot process is an anchor of trust. This means there has to be an instance in the boot process, preferably the BIOS, which can be trusted and cannot be manipulated by any attacker. The rest of the trust chain will be built on this anchor.

The next steps of the secure boot process are coupled to each other in a similar way in each step. The prior software module in the boot sequence measures (validates) the next software module in the boot sequence before executing it. The executed software module executes its own tasks, validates the following module in the boot sequence and executes it. This process is applied until the whole boot process is done. The validation of the modules can be done with hash digests or signatures of the modules.

- The BIOS validates the hypervisor and starts it.
- The hypervisor measures the boot loader of the partitions.
- The boot loader measures the OS kernel after loading it.
- The execution of the OS kernel is triggered.

A complementary technique is an authenticated boot, which is rather a passive method. This method makes use of the ECR Registers of the HSM specified in chapter 2.2.3. The partition executing an authenticated boot is obliged to update the registers with the hash values of the software components. This process results with specific register values defining the state of the software. These register values can be compared with known register values stored in the HSM and the partition software can be assured to be in a known configuration.

The result of a failed authentic boot depends on the design. A central management and/or the HSM could trigger some counteractive measure to deal with an unauthentic partition. OVERSEE could restrict access to specific key material, restrict access to I/O components or stop the partition from being executed.

The result of an authentic boot can also be used for remote attestation informing the external stakeholder of the actual configuration status. Another important attestation path would be the proof of authenticity to the application running on the partition. To assure this a trusted path between the HSM and the application has to exist. Such a path can be created with the ECR value being signed by the HSM and by a challenge/response process among the application and the HSM assuring the freshness of the signatures.

2.6 User Authentication and Authorization

Many of the OVERSEE services require the authentication of the current user of the platform and the corresponding authorization for the services. Therefore, OVERSEE offers different authentication opportunities for different kinds of applications and user types with their associated access rights.

2.6.1 User Model of OVERSEE

The OVERSEE platform has to cope with a lot of persons getting in touch with the platform in order to use some features of the platform. Obviously, not all persons sharing the same access rights. Therefore, the following user roles with their corresponding access rights are defined for the first phase of OVERSEE.

2.6.1.1 Integrator

The integrator is not a single person but the organization, which is responsible for the whole platform (e.g., building the configuration file, defining the communication policies). The integrator is hence the first contact of the OVERSEE platform to the outside world during the setup process. This process that is under the full control of the integrator will be used to store the certificate of the integrator in a secure way into the secure key and certification store of the platform. Afterwards, the public key in this certificate can be used for the verification processes (e.g., secure software load as described within section 2.7).

It is up to the integrator to decide whether or not he would like to be in charge of the authentication of the other users, as mentioned below, or if he would like to assign this task to another appropriate organization. If so, he has to store the certificate of the selected organization during the setup process in a secure way into the secure key and certification store of OVERSEE, too. For the sake of simplicity we would call the organization in charge of user authentication “authentication authority”.

Please bear in mind: Since all certificates should have a validity period and there could also be the possible requirement to withdraw certificates (e.g., in case of private key lost), there is the need for a replacement process for the authentication authority and integrator certificates during the OVERSEE usage. The integrator is responsible for the implementation of this process.

2.6.1.2 Service Staff

Service staff is a group of users which are trained to maintain the OVERSEE platform. Hence, they have access to a lot of functionality (e.g., the diagnosis services as defined within [4]), which are out of scope of other users. Obviously, there is the need for a strong authentication process. Therefore, each service staff member owns his own private key and certificate, stating which OVERSEE platform configurations he is allowed to maintain. The certificate has to be issued by the authentication authority as defined within section 2.6.1.1. The authentication authority acts therefore also as the CA (certification authority). For the sake of simplicity we will still call it authentication authority although it fulfils the obligations of a classical CA.

2.6.1.3 Owner of the Platform

The owner of the OVERSEE platform has widely access to the platform features and some configuration options. Probably the owner of the platform is also the owner of the vehicle, while this is not mandatory (e.g., in cases of business cars). From now on we would call this user type for the sake of simplicity “owner”. Since the owner could also use features with strong security concerns (e.g., software load as described in section 2.7), there is the need for a strong authentication process. Because of that the owner also has a private key and a corresponding certificate, issued by the authentication authority, and also stored in the platforms certificate store. Thus a change owner process for the platform has to be implemented by the integrator of the platform, probably online.

2.6.1.4 User of the Platform

The user of the platform would be typically the driver of the vehicle. Therefore, we will call the platform user “driver”, which would also help to avoid misunderstandings by the frequently used term user for different types of users. Since, for example in the case of business or rental cars, there could be a lot of drivers using a car, there is surely the need for an easy-to-use authentication process. Indeed, it seems not reasonable that a driver would

accept a complex authentication process each time he wants to drive a car. Hence, we provide the following simple authentication process during physical access to the platform[1]:

- The driver authenticates himself with a freely selectable password, which will be verified based on a corresponding hash value stored in the secure key and certification store of the platform.
- The authentication of the driver remains active, until he manually logs off from the platform. Hence, there is no need for entering the password each time a driver wants to drive his car. Nevertheless, there is the chance to protect the driver related information (e.g., routes in the navigation system) on driver changes, which would help to solve especially privacy concerns in shared cars, e.g., business cars.

Some special functions of OVERSEE, e.g., the assignment of flash memory devices to partitions, would require an extra proof of authenticity by re-entering the password.

New drivers could be only added to the platform while the platform owner is logged in (e.g., during a remote access session as described within [4]). For new drivers the hash value of the selected password will be stored in the secure key store of the platform. Additionally also a certificate containing the authentic public key of the driver could be stored in the certification store to provide drivers the capability to use the remote access features of the platform.

2.6.1.5 Authentication of the Platform

Achieving security is not only about being sure that the user is authentic. There is also a need to verify that the platform is authentic, especially in cases of remote access to avoid man in the middle attacks. Thus, there is the need to generate and store a private key in the secure key and certification store of the platform. This will be done during the secure setup process as described in section 2.6.1.1. The corresponding public key uniquely identifies the current platform and is provided using a certificate. This certificate has to be signed by the authentication authority and should be available for all users, for example by storing the certificate on the platform (possibly in an insecure memory area) or by distribution via a PKI of the authentication authority.

2.6.2 Authentication Methods for OVERSEE

As already mentioned in the sections above there is the need for different kinds of authentication opportunities for different types of users and use cases. The authentication methods, described within this section, will be supported by OVERSEE. The user authentication will be propagated to the generic OVERSSE components, so e.g., the security services partition allowing for example to grant or deny access to key material or protected data. Additionally the authentication could be also propagated to the partitions, which is of course only reasonable if the OS or application (in case of bare partitions) would offer a user management.

[1] There is also the option to use public key authentication to enable remote access for drivers, which should not be possible based on generally weak password authentication.

2.6.2.1 Certificate Based Online Authentication

The strongest authentication and therefore the mandatory authentication for the login of service staff and the owner of the platform is the certificate-based authentication with online access. It is very similar to the authentication used in TLS; see section about secure remote access in the D2.4 [4]. The following steps have to be fulfilled for a successfully authentication:

- The communication parties (user and platform) exchange their certificates.
- The communication parties verify the validity of the exchanged certificates. This includes verification of: Validity period, online check of CRL (Certification revocation list), certificate's signature with the public key of the authentication authority
- The communication parties authenticate each other with an challenge response mechanism, based on the private and public keys which will be implemented within the implementation phase of the OVERSEE project

Furthermore, this authentication method could be generalized, e.g., to enable authentication between different OVERSEE platforms.

2.6.2.2 Certificate Based Offline Authentication

This authentication process is similar to the process in section 2.6.2.1. The only difference is that since there is no online connection available, there is no chance to download the current CRL. Hence, it could be possible that an already withdrawn certificate would be considered valid. Since online connections cannot be assumed to be available anytime, it would be possible to login with this method for owners and service staff. Nevertheless, the information that the authentication was only conducted in offline mode will be noted in the user session. It is up to the system integrator to decide whether or not he would like to lock some platform features in case of an offline authentication. However, the platform should download the current CRL as soon as an online connection is available.

2.6.2.3 Authentication based on Extern Verification Process (optional)

This authentication is introduced for the rarely occurring situation that a specific action has to be executed by a user with an insufficient authentication, for example in the following cases:

- A service staff member has to upload an update for a buggy OVERSEE platform on the street and is not equipped with an appropriate device to conduct a certification based authentication scheme.
- All network connections to the OVERSEE platform are down and the only available interface is the USB interface to upload a software patch.

Although the current examples are only related to software load, this authentication method is not limited to this kind of use cases.

The following steps have to be run through for a successful authentication:

- The user chooses the action, which he would like to execute. Since a sufficient authentication is missing, the platform asks for his identity. Afterwards, the platform computes a challenge (based on the given identity, the selected action and a onetime value, e.g., a counter) and presents the user the following information: Unique identifier of the platform, unique identifier of the selected action, onetime value, his identity as entered as well as the calculated challenge
- The user contacts the authentication authority, for example by phone or email, and authenticates himself (this process is in the responsibility of the authentication authority). After he is authenticated he transmits the information presented by the platform. The authentication authority calculates, based on the given information, an appropriate response and transmits this response to the user.
- The user enters the response into the platform by using the HMI and if the response matches to the expected response by the platform, the user is authenticated. **Attention: The user is only authorized to execute this action one time per authentication.**

Since this authentication method is optional no further details about the mentioned challenge response mechanism are given here.

Please keep in mind that an authentication as described in this section inherently offers a lot of security weakness (e.g., it depends on the security of the authentication process between user and authentication authority and is only a one way authentication). Therefore, this method is no part of the generic OVERSEE implementation and would also not be recommended. Nevertheless, it is up to the integrator to decide whether or not he would need it. However, this authentication method should only be available during direct physical access to the platform.

2.6.2.4 Pre-Shared Key Based Authentication

This authentication process is very simple, since it is based on a hash value of a pre-shared key which has been already stored in the secure key and certification store of the OVERSEE platform, see section 2.6.1.4. Since it is only a one-way authentication with strong security weaknesses (e.g., the security level strongly depends on the selected passwords), it should only be used to authenticate drivers with physical access to the platform. The following steps have to be run through for a successful authentication:

- The driver enters his password into the platform
- The platform calculates the hash value of the entered password
- If the hash value matches the stored hash value in the secure key and certification store the authentication process is successfully finished

2.6.3 Extended Authorization Capabilities

As already stated the different groups of users (integrator, service staff, owner and driver) owning different access rights for the platforms functionality. Beside this simple assignments based on user groups, there is also the capability to use additional fields in the users certificates, to enable a more precise assignment of access rights on a per user base. This

option will be especially important for the service staff and owner group, e.g., to support different groups of service staff or allow remote feature activation mechanisms per owner.

2.7 Secure Software Load

Secure software load is the capability of a system to replace or upgrade its own software components during usage in a secure way. Secure means that only authorized users are able to start the software load process and that only proper new software could be loaded and installed. Software as referred to in this section is either a software component of the OVERSEE implementation (e.g., a new version of the OVERSEE management application) or the whole content of a partition (including applications, data and − if any − the operating system). It has to be stressed, that OVERSEE ensures security on partition level and not on application level. Hence, it is up to the responsible organization for a partition to provide an own application load concept if reasonable. Anyway, the responsible organizations are invited to reuse parts of the secure software load process as described in this section and be part of the generic OVERSEE implementation.

2.7.1 Requirements for a Secure Software Load

The following requirements have to be fulfilled to ensure that downloaded software can be installed without doubts. Obviously, the installation process can only be started if the vehicle is not moving and the platform is not running any critical service, since this could lead to safety critical implications.

2.7.1.1 Authenticity and Authorization of Installing Person

Two groups of persons should be able to download software and start or at least prepare the installation process: The platform owner and authorized service staff. The software download process could be either started with direct access to the platform or during a remote access session as described in the dedicated section in [4]. The following authentication methods are supported:

- Certificate Based Online Authentication, as defined in section 2.6.2.1
- Certificate Based Offline Authentication, as defined in section 2.6.2.2
- Authentication based on external verification process (if implemented), as defined in section 2.6.2.3

2.7.1.2 Integrity Check of the Downloaded Software and Configuration File

After a successful authentication of the installing user the next step is to check the integrity of the downloaded software package. The package consists of the following parts:

- Binary of the new software component
- Configuration file for the installation process and compatibility checks as described in section 2.7.1.3 in XML format

- Signature over the binary file and the configuration file, generated with the private key of the platform integrator

The integrity check simply verifies the signature with the public key of the platform integrator stored in the secure key and certification store of the platform to verify the integrity of the downloaded software and configuration file.

2.7.1.3 Compatibility Checks

To ensure that only appropriate software versions will be installed on the platform, the configuration file at least states the following information:

No	Field Name	Description	Applied Checks
1	ID	A unique ID of the software package	Check installation log if the id occurs the first time
2	Name	The name of the software component contained in the package	
3	Version	Version number of the software component	
4	Integrator	Unique identifier of the integrator responsible for the platform, corresponding to the public key which was used to verify the signature	Check if integrator matches
5	Component ID	Unique ID of the component as assigned by the integrator of the platform that should be updated	Check if component ID matches to an installed component
6	Old Versions	List or range of old version numbers of the component, which could be updated with the current package	Check if the current version matches to the updateable versions

Table 26: Content of software download configuration file

Since the configuration file is in XML format it could be easily adapted to the requirements of the platform integrator.

3 Specification of the Whole Virtualization Concept

Partitioned software architectures can represent the future of secure systems. They have evolved to fulfil security and avionics requirements where predictability is extremely important. The separation kernel proposed in [22] established a combination of hardware and software to allow multiple functions to be performed on a common set of physical resources without interference. The MILS (Multiple Independent Levels of Security and Safety) initiative is a joint research effort between academia, industry, and government to develop and implement a high-assurance, real-time architecture for embedded systems. The technical foundation adopted for the so-called MILS architecture is a separation kernel. Also, the ARINC-653 [16] standard uses these principles to define a baseline operating environment for application software used within Integrated Modular Avionics (IMA), based on a partitioned architecture.

Virtual machine technology can be considered the most secure and efficient way to build partitioned systems. A virtual machine (VM) is a software implementation of a machine (computer) that executes programs like a real machine. Hypervisor (also known as virtual machine monitor VMM [23]) is a small layer of software (or a combination of software/hardware) that enables to run several independent execution environments or partitions in a single computer. The key difference between hypervisor technology and other kind of virtualisation (such as java virtual machine or software emulation) is the performance. In bare-machine hypervisors the overhead can be very low maintaining the throughput of the virtual machines very close to the native hardware.

Hypervisor is a new and promising technology, but has to be adapted and customized to the requirements of the target application. The low overhead and the reduced size of the hypervisor can be considered as an appropriated solution to achieve secure systems if it is designed following strict design criteria to meet security requirements. Its correctness can be sufficient to ensure the security of the system as a whole or, at least, the security of a set of trusted partitions. In a partitioned system, the partitions can accommodate different kinds of applications: real-time, trusted, non-trusted, etc. As consequence, the partition's operating system can be customised to provide the specific services to its applications.

A hypervisor is in charge of virtualisation services to partitions. It is executed in supervisor processor mode and virtualises the CPU, memory, interrupts and some specific peripherals.

3.1 Partitions

A partition is an execution environment managed by the hypervisor that uses the virtualised services. Each partition consists of one or more concurrent processes (implemented by the operating system of each partition), sharing access to processor resources based upon the requirements of the application. The partition code can be:

- An application compiled to be executed on a bare-machine (bare-application)
- A real-time operating system (or runtime support) and its applications (i.e. RTEMS, POSIX PSE compliant, ARINC653 compliant)
- A general purpose operating system and its applications (i.e. Linux, Android, …)

Partitions need to be virtualised to be executed on top of a hypervisor. Depending on the type of execution environment, the virtualisation implications in each case can represent low or significant efforts. A RTOS with a very well defined HAL (Hardware Access Layer) can require low effort to change the HAL services accessing directly to the hardware by hypervisor services. A Linux para-virtualisation could involve much more efforts. However, the Linux kernel has evolved to provide para-virtualisation services. In this case, the efforts required can be significantly reduced.

The hypervisor approach permits to define partitions with different levels of security. Some partitions can have limited rights but others can access to special services as reset, start, stop, etc., other partitions.

3.1.1 Virtualized Resources

The hypervisor approach is based on the resource virtualisation to the partitions. So, partitions are virtual machines that access to the virtualised resources through the services provided by the hypervisor. The basic resources can be classified as:

- Processor management: includes the processor and registers
- Clock and timers: includes the clock device and timers
- Interrupts: includes the interrupt lines and can add additional new interrupts (virtual interrupts)
- Memory regions: areas of memory that can be allocated exclusively or shared to partitions. This also can include I/O memory areas
- Basic peripherals: as serial lines or other basic devices

This vision is complemented by the dedicated devices approach. This approach permits to define devices that are exclusively and directly managed by a specific partition. These devices are not virtualised by the hypervisor and the partition has the responsibility of their management. In this category are the network interface, disk, etc.

3.1.2 Configuration and Deployment Overview

A significant change in this approach that arises from partitioned system development is the system architect role. He or she has the responsibility of the system definition and configuration. It implies the definition of the partitions to be executed and the resources allocated to each partition. This specification is detailed in the configuration vector.

It contains the information as: memory requirements, processor sharing, peripherals, health monitoring actions, etc.

- Memory requirements: The amount of physical memory available in the board and the memory allocated to each partition.
- Processor sharing: How the processor is allocated to each partition by specifying the scheduling policy or plan.
- Native peripherals: Peripherals that are not virtualised and can be used by one partition. The I/O port ranges and the interrupt line if any.

- Health monitoring: How the detected error are managed: direct action, delivered to the offending partition, create a log entry, etc.
- Inter-partition communication: The ports that each partition can use and the channels that link the source and destination ports.

Since the configuration vector defines the resources allocated to each partition, it represents a contract between the system architect and the partition developers.

The configuration vector has to be validated and integrated in the system deployment. As it is a contract, the hypervisor has to grant that the allocated resources are used as specified. The hypervisor has to guarantee that the resources are used by the right partitions.

3.2 Concepts

This section introduces the basic concepts defined in Common Criteria to fulfil the security requirements which are used in the SKPP.

The TOE (Target Of Evaluation) is a virtualisation layer designed to instantiate and manage partitions that serve to host custom applications. The TOE manages access to memory, devices, communication resources and processor resources to ensure that partitions are completely isolated and can interact only with its resources allocated by the System Architect in the configuration vector.

The System Architect creates one or several static configuration vectors that define the partitions of the system, the subjects and resources. The configuration vector also permits to specify the security functionalities (TSF) related to the partitions and resources.

The virtualisation layer enforces that each virtualised and exported resource can be accessed by a partition at a time. To achieve this goal, the TSF ensures that partitions are executed according to a scheduling policy defined in the configuration vector. This plan can be analysed off-line to guarantee the temporal constraints of the applications through a schedulability analysis.

For resources such as memory, which do not require mutual exclusion to the whole, the TSF provides full isolation by allocating physically distinct portions of the resource to different partitions. TSF ensures the spatial isolation of its internal resources. Subjects, and resources made available to subjects by the TSF, are called *exported resources*.

The Partitioned Information Flow Policy (PIFP) defines the rules for isolation granted by the TOE. It defines the authorisations for information flow between partitions and between subjects (application using a resource) and exported resources (concrete resources used by the applications). An information flow is defined as a <partition/subject, partition/exported resource, mode> triplet. Note that the exported resource may be another subject. All the information flows have to be specified in the configuration vector. By default, no information flow between partitions or between subjects and exported resources is allowed.

3.2.1 Subjects, Objects and Privileges

Based on the Common Criteria definitions:

- **Subject**: active entity in the TOE that performs operations on objects. It can be categorised in two types: system and user.
- **Exported resources**: passive entity in the TOE, that contains or receives information, and upon which subjects perform operations.
- **Operation Mode (on a resource) specific type of action performed by a subject on an exported resource.**

3.2.2 Subject Identification

The virtualisation layer manages a partition as main active entity. Processes inside of a partition are handled internally and the virtualisation layer does not know their existence. It is the partition that is responsible of its management. Any operation performed by any of the internal active elements of a partition is seen as a partition operation. Based on this approach, the set S of subjects is formed by all the partitions defined in the configuration vector.

3.2.3 Exported Resource Identification

The virtualisation layer has the knowledge of the system via the configuration vector. In this configuration vector, all the subjects, exported resources and operations have to be defined.

The list of exported resources is:

- CPU registers: CPU and internal registers
- CPU time: System clock
- FPU: Floating point unit
- Traps: Processor traps
- Timers: System timers
- Memory layout: Board available memory defining the allocations and sizes
- Memory area: Memory areas allocated to partitions specifying the allocation and size
- Shared Memory area: Memory areas shared between partitions.
- Memory block area: Memory areas that are handled as devices
- Scheduling Policy: Policy to schedule partitions
- Channels: Links between partition ports. Two types of channels can be specified: Queuing and Sampling channels. Specific parameters associated to a channel can be: maximum meassage length, maximum number of messages, who is the reader/writer, etc.

- Devices: Specific devices as UART, VGA, etc.
- I/O Ports: Hardware IO ports.
- Interrupts: Hardware interrupts

3.2.4 Partitions and the Partitioned Information Flow Policy

The identified operations on exported resources are:

- **Read**: A subject can read from the exported resource.
- **Write**: A subject can write to the exported resource.
- **Run**: A subject is only allowed to use the exported resources during the run operation.

The TOE provides partitions as abstraction implemented by the TSF. It is guest OS personality that is in charge of managing the internal subjects (threads or tasks or processes) that are not visible from the virtualisation layer. From this point of view, it is assumed the Partition Abstraction policy:

> **The subjects in a partition have homogeneous requirements for access, on a per-partition basis, to exported resources.**

It is responsibility of the guest OS to define another policy. For instance, a guest OS could define a Least Privilege Abstraction which assumes that the subjects in a partition have heterogeneous requirements for access to exported resources. In this case, the guest OS could restrict the operations defined in the other policy to some internal subjects. Privileges of each resource are specified by means of an access matrix.

3.2.5 Auditable Events

Auditable events provide information to the security auditing to recognise, record, store, and analyse information related to security relevant activities. The resulting audit events can be examined to determine which security relevant activities took place and who is responsible for them.

Audit events can be generated by the virtualisation layer as result of the

- internal changes: secure/unsecure state, initialisation completed, partition started/stopped, etc.
- exceptions captured by the Health Monitor: overflow, division by zero, illegal memory address, etc.

These audit events are logged and can be analysed by a partition in order to perform an attestation. Event records will contain the following information:

- eventId: the type of the event
- initiator: virtualisation layer or partition
- partitionId the identifier of the partition (if not a system event)
- info: event specific information

- timeStamp: time at which the event was detected

3.3 Secure States

3.3.1 Concepts

The definition of the secure state is based on two separate properties: (A) that the TSF is capable of enforcing the security policy (i.e., its own data and mechanisms are intact); and (B) that exported resources are correctly separated (e.g., application data, and related descendants and copies, are associated with the correct data).

Property (A) states that the "secure state" is strongly related to the integrity and coherence of the internal data and mechanisms. Internal data can be considered as:

- Configuration vector as binary representation of the configuration file used to define the system that has been validated, compiled and included in the final system container. During execution, this configuration vector resides in the hypervisor (kernel) address space (not accessible by subjects) in a memory area that is write-protected. Additionally, a digest (cryptographic hash function) is applied to the configuration vector, which is added to it. At any moment, the TOE can compute the digest of the configuration vector and validate its integrity.
- Internal variables: state of the TOE. TOE status refers to the internal variable that maintains the execution status: current partition under execution, current plan, current slot and current time. The coherence of these variables is fundamental in the TOE operation.
- The processor registers: MMU registers , interrupt vector, mode processor status
- Channels: the consistency of the channel data structures can be determine with respect to the maximum values (message length and number of messages) defined in the configuration vector.
- Stacks: The TOE maintains one stack for its own TOE operations and one stack per partition, which is used when the TOE executes a hypercall for a specific partition. The limits of these stacks can be validated. It is important to note that each partition maintains its own stack in the user space when the partition is executing operations at user level.

Property (B) is related to the isolation properties (spatial and temporal) of subjects. The TOE will be designed so that the individual effects of operations that violate the policy are privileged operations (operations over virtualised resources) or by means of hypercalls with not allowed parameters (i.e. reset the system by a subject not authorised). It means that privilege instructions that can compromise the isolation (for instance change the interrupt mask) only can be performed by the TOE. In order to change it, a partition has to use the virtualised services and not access directly to the hardware.

In the first case, the forbidden operation will generate a trap that will be captured by the TOE and generate a Health Monitor event which involves an HM action with the goal of maintain the TOE in the ”secure state” (i.e. the subject can be halted (disabled) or restarted according to the action defined in the configuration vector. In the second case, the hypercall

with non-allowed parameters, the TOE will perform an exhaustive validation of the parameters according the configuration vector and refuse the operation (returns a code error in the hypercall to the subject invoking the hypercall).

When the conditions stated previously cannot be validated, the TOE will be in an "insecure state". The following situations can determine that the TOE is in an "insecure state":

- Configuration vector pollution. The digest of the configuration vector does not match with the correct value.
- TOE code pollution. The digest of the TOE code does not match with the correct value.
- Deviation of the internal state.
- Access to non-exported resources for a partition. A partition can perform an operation to an exported resource that has not been defined in the configuration vector. (How can it be detected?). Note that if a partition requests an operation on a non-exported resource the hypercall should return a code error.
- Limits exceeds. Stacks or channels data structures exceeds the limit values established in the configuration vector.
- Underlying hardware: clock, timers, memory protection mechanisms, I/O protection mechanisms, FPU protection mechanisms.

Any of these situations determine that the TOE is not in a "secure state". In these cases, it is not possible to change to a "secure state" and the system has to be reset.

3.3.2 Operations

Some aspects have to be considered:

- TOE shall be non pre-emptable. When any of the entry-points is invoked, it is executed with disabled interrupts returning the control to a partition.
- All exported resources are defined in the configuration vector.
- Only the hypervisor can access processor registers and virtualised services.
- Internal code of partitions is not relevant from the hypervisor point of view

Additionally, we assume that the underlying hardware is trusted. It means that the internal processor registers will work properly if they are used in the correct way. For the temporal and spatial isolation purposes, it is assumed:

- The access to the processor registers is only allowed when the processor is in privileged mode. The processor mode can set/unset by accessing the control processor status (PMS).
- The MMU controls the memory areas and will raise an exception when an instruction tries to read or write in a memory area out of range.
- A timer is used by XtratuM to control the slot duration.

- The interrupt vector is handled exclusively by XtratuM. It can only be accessed or modified when the processor is in privileged mode.

As result of the previous analysis the state of the TOE could be evaluated at different levels using different tests:

- **Basic test**: Basic test relies with basic properties of the hypervisor and the trust enforcement from the trusted hardware. It includes:
  - Validation of the internal variables related to the TOE state
  - Processor registers
  - Stack limits
  - Monotonic clock

  This test requires few computation resources. It is validated each time the TOE performs a partition context switch.
- **Abstract machine test**: In general the AMT refers to the proper operation of the hardware platform on which a TOE is running. This test permits to consider that the underlying hardware is trusted. It includes:
  - Timers test
  - Protection mechanisms test: MMU, privileged operation, I/O protection, FPU control.
  - Memory Read and Write: This test can read/write/read portions of memory to ensure the integrity of the values written remain unchanged.
  - Memory Separation and Protection: to ensure that user space programs cannot read and write to areas of memory that is protected.
- **Self-tests**: Self tests are related with the self evaluation of the TSF with respect to some expected correct operation. It includes:
  - Stack limits: TOE and partition's stacks.
  - Configuration vector (perform a digest of the current configuration vector and compare it with the deployed digest).
  - TOE code (perform a digest of the XtratuM code and compare it with the deployed digest).
  - Channel limits evaluation
  - Partition code pollution evaluation. This is an operation that should be performed by each partition.

3.4 TOE Attestation

TOE attestation is referred to the ability to access to the events that has been generated during the TOE execution. These events are logged in an internal data structure and can be accessed by a partition with the appropriated rights (system partition). Logged events can be:

- Auditable events (described below). They are always logged.
- Health Monitor events. Each event type should be logged or not. This decision corresponds to the designer and can be specified in the system configuration.
- Traces and Logs generated by partitions.

3.4.1 Health Monitor Events

Health Monitor (HM) defines the action to be done by the TOE when an error or fault is detected. This action should be very clear and precise in order to limit the effects of the error and avoid its propagation. The purpose of the HM is to discover the errors at an early stage and try to solve or confine the faulting subsystem in order to avoid or reduce the possible consequences.

The event occurrence, which implies, the action execution should be logged or not according to the nature and importance of the detected error. HM is executed as result of a HM event occurrence. The following scenarios can raise a HM event:

- An exception has been raised by the CPU. The exception handler generates the associated HM event.
- A native interrupt has been received and the temporal or spatial properties are not validated.
- A trap has been received and the temporal or spatial properties are not validated.
- A partition detects an abnormal internal situation and raises a HM event. For instance, the operating system inside of a partition detects that the application is corrupted.

The HM event occurrence is the manifestation of an error. The TOE reacts to the error providing a simple set of predefined actions to be done when it is detected.

As example, the following table shows some of the errors that can be handled by the TOE. Some of them can be generated by the TOE or partition operation and others can only be generated by the Partition. The first block involves the analysis of who is responsible for the error and the action can be different depending on the generator.

No	TOE and partition operation
1	WRITE_ERROR
2	INST_ACC_EXCEPTION
3	ILLEGAL_INST
4	PRIVILEGED_INST
5	FP_DISABLED
6	CP_DISABLED
7	REG_HW_ERROR
8	MEM_ADDR_NOT_ALIG
9	FP_EXCEPTION
10	DATA_ACC_EXCEPTION
11	TAG_OVERFLOW
12	DIVIDE_EXCEPTION

Table 27: Captured errors, TOE and partition operation

No	Partition operation
1	MEM_PROTECTION
2	PARTITION_INTEGRITY
3	PARTITION_UNRECOV.
4	OVERRUN
5	SCHED_ERROR
6	INTERNAL_ERROR
7	UNEXPECTED_TRAP

Table 28: Captured errors, Partition operation

Actions have to be simple and precise in order to limit the effects of the error and not to introduce temporal interference with other partitions. If an action could require a significant amount of time, a partition could impact on the temporal isolation by generating continuously an error. Some of the predefined actions are detailed in the next table.

No	Predefined actions
1	Cold_Reset
2	Warm_Reset
3	Ignore
4	Propagate (to the partition)
5	Stop_Partition
6	Suspend_Partition
7	Restart_Partition

Table 29: Predefined actions

3.4.2 Traces and Logs

The TOE shall provide a mechanism to store and retrieve the traces generated by partitions. Traces can be used for debugging, during the development phase of the application, but also to log relevant events or states during the production phase.

In order to enforce resource isolation, each partition (as well as TOE) will dedicated trace log streams to store the trace messages generated by each partition and TOE. Trace streams are stored in buffers (RAM or FLASH). Only system partitions should read from a trace stream.

4 OVERSEE Firewall

Since a main topic of OVERSEE is the secure integration of a wide range of communication means, one of the main security issues is to isolate the different communication interfaces from each other and avoid intrusions through this interfaces. The current section describes the communication paths of OVERSEE, as defined within [18] and continues the security considerations towards these paths as mentioned in [4]. Following the OVERSEE firewall will be specified including a role model for policy specification and the policy language.

4.1 Communication Paths and Interfaces

Within [18] the communication capabilities of OVERSEE towards the environmental networks and the services towards the applications running within the OVERSEE runtime environments were defined. Based on the separate view per communication resource in that deliverable in Figure 3 all communication paths within OVERSEE will be presented.

For Figure 3 the following meanings of symbols and colours apply:

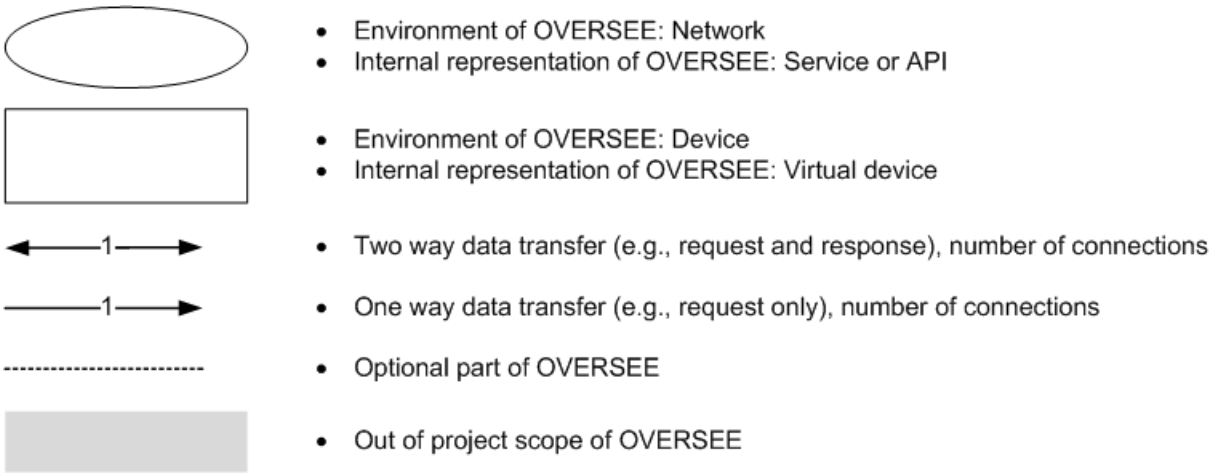

Figure 2: Key for OVERSEE communication paths

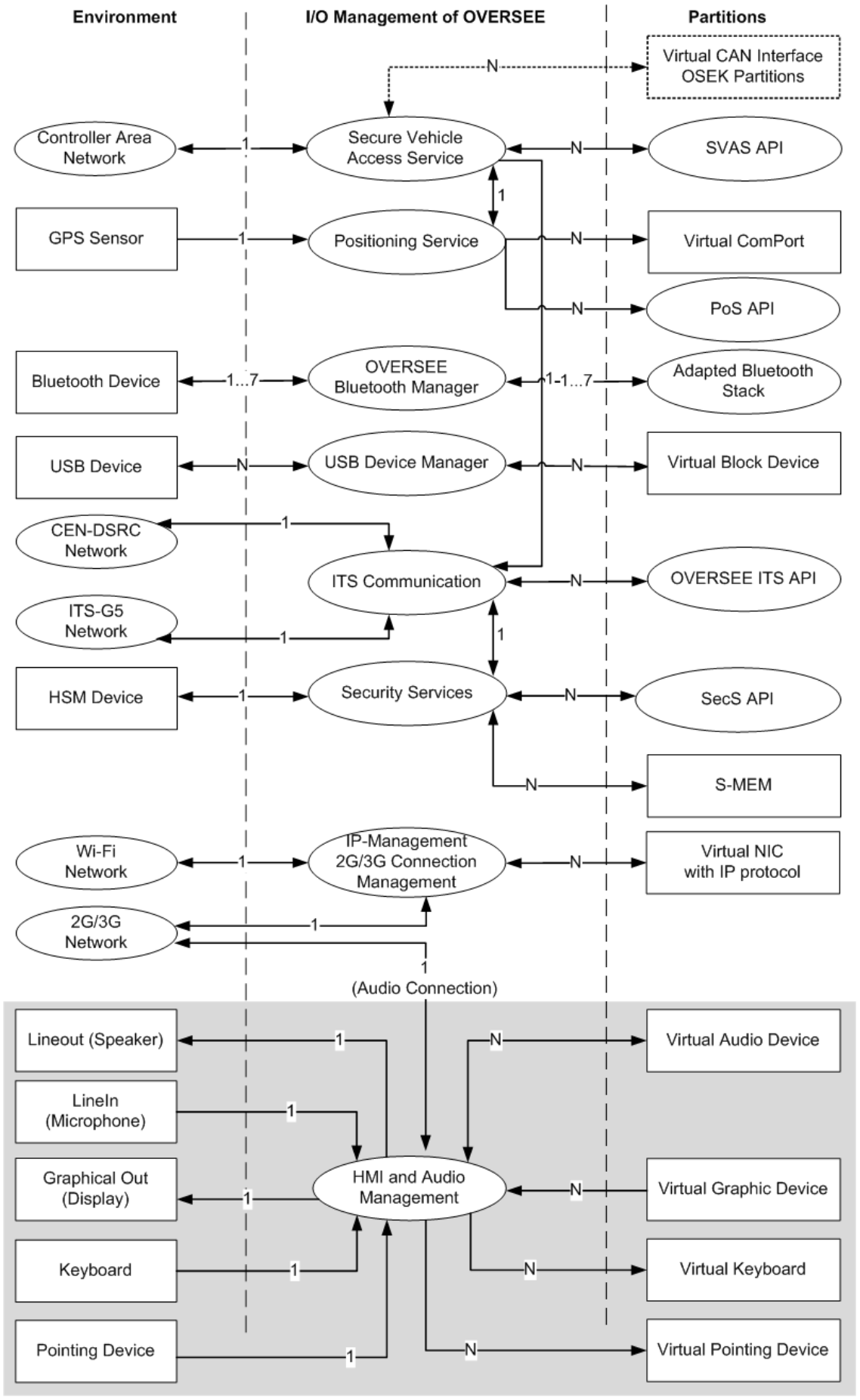

Figure 3: OVERSEE communication paths

4.1.1 Security Methods Concerning the Communication Paths to be Considered for the OVERSEE Firewall

The following table summarises the security methods, belonging to the communication paths as depicted in [4], to be considered for the OVERSEE firewall.

No	Connection, Interface, Device, Service	Description of security method	Consequence for firewall implementation
1	CAN ⇔ SVAS	Restrict read / write access on a message based policy, according to white lists defined by the OEM	Part of the OEM specific implementation part of SVAS, hence out of OVERSEE firewall scope
2	SVAS	Restrict access (read/write) on a data object based policy per partition, according to the rules defined by the OEM	Set of rules will be part of the OVERSEE firewall configuration
3	SVAS ⇔ SVAS API	Restrict access to SVAS per partition	Rule will be part of the OVERSEE firewall configuration
4	OSEK virtual CAN connection	Only optional, hence not considered in this document	
5	Positioning service	Restrict access to positioning service per partition	Rule will be part of the OVERSEE firewall configuration
6	Positioning service ⇔ SVAS	Restrict access according to rules defined in D2.4 [4]	Fixed part of the generic OVERSEE implementation, hence out of scope of the OVERSEE firewall
7	Bluetooth connection	Restrict capability to use Bluetooth connections per partition	Rule will be part of the OVERSEE firewall configuration
8	USB device	Restrict capability to assign USB devices per partition	Rule will be part of the OVERSEE firewall configuration
9	ITS communication	Restrict delivery of CAM messages per partition	Rule will be part of the OVERSEE firewall configuration
10		Restrict access to the LDM (if any) per partition	Rule will be part of the OVERSEE firewall configuration

11		Restrict delivery of DENM messages per partition	Rule will be part of the OVERSEE firewall configuration
12		Restrict delivery of DSRC messages per partition	Rule will be part of the OVERSEE firewall configuration
13		Restrict write access to parameters for CAMs per partition	Set of rules will be part of the OVERSEE firewall configuration
14		Restrict sending of DENM messages per partition	Rule will be part of the OVERSEE firewall configuration
15		Restrict sending of DSRC messages per partition	Rule will be part of the OVERSEE firewall configuration
16	ITS communication ⇔ OVERSEE ITS API	Restrict access to ITS communication service per partition	Rule will be part of the OVERSEE firewall configuration
17	ITS communication ⇔ SVAS	Read only access for a specified set of data objects by the ITS communication	Set of rules will be part of the OVERSEE firewall configuration
18	Security services	Restrict access to security services (including S-MEM) per partition	Rule will be part of the OVERSEE firewall configuration
19	IP-Management	Restrict access to IP connections based on the connection type per partition	Set of rules will be part of the OVERSEE firewall configuration
20		Do not route IP packets between the virtual IP connections towards the partitions	Fixed part of the generic OVERSEE implementation, hence out of scope for OVERSEE firewall
21		Restrict IP traffic based on the used ports per partition	Set of rules will be part of the OVERSEE firewall configuration
22	IP-Management ⇔ Virtual NICs in partitions	Restrict access to IP management per partition	Rule will be part of the OVERSEE firewall configuration
23	HMI and Audio Management	Restrict use and setup of 2G/3G voice connections per partition	Rule will be part of the OVERSEE firewall configuration

24		Restrict access to audio output and input per partition	Rule will be part of the OVERSEE firewall configuration
25		Restrict access to graphical output per partition	Rule will be part of the OVERSEE firewall configuration
26		Restrict access to keyboard and pointing device input per partition	Rule will be part of the OVERSEE firewall configuration

Table 30: Security issues to be handled by the OVERSEE firewall

Please consider the residual risks and counter measurements, which are out of project scope, mentioned in D2.4 [4], e.g., the suggested application layer firewall for IP connections.

4.2 Specification of the OVERSEE firewall

In section 4.1.1 we summarised the necessary security measurements per communication path and the consequences for the OVERSEE firewall. As stated for most of the measurements there will be a rule or a set of rules in the firewall configuration of OVERSEE to reduce the risk of misuse of the connections. To be able to define a holistic security concept, all the rules will be specified within one configuration file in XML format for each platform configuration. The XML file has to be signed by the platform integrator. Only configuration files with verified signature will be processed by the OVERSEE platform.

During processing of the configuration file four configuration files will be build out of the original configuration file, corresponding to specific parts of the OVERSEE firewall:

- Parts of the XtratuM configuration file [17], describing the available ports to specified channels per partition (methods number: 3,5,7,8,16,18,22,23,24,25,26)
- Firewall configuration file for SVAS (methods number: 2,17)
- Firewall configuration file for ITS communication (methods number: 9,10,11,12,13,14,15)
- Firewall configuration file for IP communication (methods number: 19,21)

Since the used hypervisor XtratuM is able to guarantee that partitions only access channels if they have a corresponding port configured in the XtratuM configuration file, the restriction of access to the services is implemented by the first configuration file easily and secure. The other three configuration files belonging to separate parts of the OVERSEE firewall, executed within the corresponding domain, are the following:

- SVAS access control module, executed within the SVAS service and processing the firewall configuration file for SVSA
- ITS access control module, executed within the ITS communication service and processing the firewall configuration file for ITS communication

- IP filter module, executed within the IP management service and processing the firewall configuration file for IP communication

How to implement the access control and filter modules in the services will be defined within the implementation phase of the generic OVERSEE components in WP3.

4.2.1 Role Model for Definition of Firewall Policies

There are two roles for the definition of firewall policies in the context of OVERSEE:

- OEM: Responsible for the definition of white lists for CAN read/write and the set of rules concerning the access of data objects within the SVAS
- Integrator: Responsible for all other rules and the reliable integration of the rules concerning SVAS access as defined by the OEM into the OVERSEE firewall configuration file

During the implementation phase of OVERSEE it will be determined if there is also the need for dynamic firewall policies, which could be selected based on the current status of the platform. However, the responsibilities for the definition of the applicable rules while be not affected by this decision.

4.2.2 Policy language for the OVERSEE firewall

As already mentioned, the firewall configuration file will be in XML format. The following DTD (document type definition) represents the current reflections concerning the definition of the rules as listed in section 4.1.1. Anyway, the policy language will surely be extended and adjusted during the implementation phase and also beyond the current project in real world implementations; luckily this is easily feasible due to the XML format of the configuration file.

OVERSEE_firewal_policy_language.dtd, Beta, Version 0.1:

```
<!ELEMENT ComRules (Services, SVAS, IP, ITS) >
<!ELEMENT Services (Service)* >
<!ELEMENT Service (Partition)*>
<!ATTLIST Service
          name CDATA #REQUIRED
>
<!ELEMENT Partition EMPTY>
<!ATTLIST Partition
          name CDATA          #REQUIRED
          read (true|false)   #IMPLIED
          write (true|false)  #IMPLIED
>
<!ELEMENT SVAS (DataObject)*>
<!ELEMENT DataObject (Partition|IntConnection)*>
<!ATTLIST DataObject
          name CDATA #REQUIRED
>
<!ELEMENT IntConnection EMPTY>
<!ATTLIST IntConnection
```

```
          name CDATA          #REQUIRED
          read (true|false)   #IMPLIED
          write (true|false)  #IMPLIED
>
<!ELEMENT IP (ConTyp|FWRule)*>
<!ELEMENT ConTyp (Partition)*>
<!ATTLIST ConType
          name (WiFi|2G3G)    #REQUIRED
>
<!ELEMENT FWRule (Partition)*>
<!ATTLIST FWRule
          portnumber CDATA    #REQUIRED
>
<!ELEMENT ITS (CAM|LDM|DENM|DSRC|ParamCAM)*>
<!ELEMENT CAM (Partition)*>
<!ELEMENT LDM (Partition)*>
<!ELEMENT DENM (Partition)*>
<!ELEMENT DSRC (Partition)*>
<!ELEMENT ParamCAM (Partition)*>
<!ATTLIST ParamCAM
          name CDATA          #REQUIRED
```

5 Secure Assurance of Prioritized Resource Access

Parallel execution of multiple applications sharing common resources is one of the main features of OVERSEE. The executed applications could be from different domains (e.g., infotainment, comfort, ITS) and some of them are safety relevant.

A short example of an ITS application, in this case a simplified[2] version of eCall as described within [1], is shown in the following:

- The vehicle internal eCall implementation detects sensor values that lead to the assumption that an accident has occurred (e.g., the airbag or some crash sensors have been activated). As an alternative the vehicle driver activates the eCall feature manually, because of an emergency situation (e.g., heart disease or feeling the risk of loss of consciousness).
- The eCall application presents information that it connects to the next PSAP (Public Safety Answering Point) and a cancel button via the HMI to the driver.
- The internal eCall implementation establishes a mobile phone voice connection to the next PSAP and transmitting specified information (among others positioning, direction, status of airbags) in-band to the PSAP. Afterwards, the voice connection to the PSAP will be forwarded to the vehicle driver.

The simple (compared to complex ITS applications as intersection collision warning) use case already shows some safety relevant issues concerning shared resources:

- Although other applications are executed on the OVERSEE platform it has to be ensured that the partition serving the eCall application is regularly invoked, to check if there are indications for an accident.
- If an accident is detected or the eCall feature is activated manually the application has to inform the driver. Hence, exclusive access to the HMI has to be assigned to the partition serving the eCall application.
- Since there is the necessity of a voice connection between driver and PSAP the eCall application has to interrupt all currently established voice connections. Hence, exclusive access to the 2G/3G and audio management module of OVERSEE has to be assigned to the partition serving the eCall application.

This list of safety relevant issues, while it is not exhaustive, already leads to some first security concerns, related to the mentioned safety relevant issues:

- How to avoid DoS (Denial of Service) attacks by applications avoiding the invocation of, in this case, the partition serving the eCall application.
- How to avoid that malicious applications masquerade themselves (to be exactly their hosting partition) as eCall applications (to be exactly the partition serving the eCall

[2] Although this description is simplified, compared to the use case description of eCall in [1], it fits to the eCall implementation approach recommended by the eSaftey Forum [20].

application) and hence gathering exclusive access (e.g., on the 2G/3G connection management).

Therefore, secure assurance of prioritized access to shared resources will be a feature of OVERSEE. Additionally, some resources offer advanced resource sharing techniques (e.g., sharing of data transmission rate by applications with QoS (Quality of Service) approaches). The secure assurance of advanced resource sharing techniques will be also part of the OVERSEE implementation.

5.1 Classes of Resources Concerning Prioritized Resource Access

As already mentioned, resources could be classified concerning prioritized resource access into two groups:

5.1.1 Time shared resources

The use of resources within this class requires exclusive access to the resource. Hence, the only possible solution is to share access to these resources over time. This could happen in a cyclic manner, granting access to the resources to consumers on a regular time base or dynamically according to the needs of the current costumers (e.g., applications).

5.1.2 Resources capacity shared resources

The resources within this class offer the capability to split of the capacity of the resource among the consumers. This could be for example the data transmission with configuration of adaptive rates offered by a network connection or the available memory. Often these systems are also able to assign the available resource capacity dynamically according to the current requirements of the consumers, referred to as QoS information for network connections. For the sake of simplicity we will call this techniques "sharing of resource capacity".

Nevertheless, as a fact it has to be noticed that even resources offering sharing of resource capacity are mostly based on other components, which are only able to offer sharing over time (e.g., the SVAS using CAN which could indeed transport only one message at the same time). Since the consequences of these relations are typically known they are already considered in the capacity rates of the depending resources. We will hence consider issues related to this fact only if necessary in a security point of view.

5.1.3 Classification of OVERSEE Resources Concerning Prioritized Resource Access

In this section we will classify the resources of OVERSEE according to the two different groups, defined above. Please notice that resources are not the same as communication paths. Indeed, sometimes more than one resource builds a communication path and hence has to be evaluated separately. Obviously, parts of the communication paths, which are executed in partitions do not need resource sharing (keep in mind OVERSEE works on partition and not application level). This also applies to read only resources. Furthermore, we consider also resources, which are not related to communication (e.g., CPU time).

Resource	Shared over time	Shared resource capacity	Relevance for secure assurance of prioritized resource access in OVERSEE
Controller Area Network Interface	X		Implementation out of scope, will not be considered
SVAS		X	Please see section 5.1.4 for more details on the shared resource capacity
SVAS to CAN output queue	Implement the queue in a manner that reflects the prioritization in CAN.		
Data Objects in SVAS	X		Demand for prioritized exclusive access per data object
GPS Sensor	Read only therefore no issue		
Positioning Service	Read only therefore no issue		
Bluetooth Device	Since the user selects to which partition the Bluetooth device should be assigned, there is no issue on prioritization.[3]		
USB Device	Since the user selects to which partition the USB device should be assigned, there is no issue on prioritization.		
CEN DSRC Interface	X		Implementation out of scope, will not be considered
ITS-G5 Network Interface	X		Implementation of prioritization within the ITS-G5 network is out of scope.
ITS Communication		X	The considerations of prioritization have to be on the level of write processes for parameters, used for the CAMs, and send of DENM and DSRC messages.
Parameters for CAMs in ITS communication	X		Demand for prioritized exclusive access per parameter
Send of DENM	X		Demand for prioritized exclusive access
Send of DSRC messages	X		Demand for prioritized exclusive access
HSM Device	X		Implementation out of scope, will not be considered
Security Services	X		Demand for prioritized exclusive access[4]
WiFi Network Interface	X		Implementation out of scope, will not be considered
2G/G3 Network Interface	X		Implementation out of scope, will not be considered
IP connection		X	Demand for prioritized resource capacity sharing

[3] The communication within the Bluetooth network will be under the responsibility of the used Bluetooth module.

[4] Additional measures in the implementation of the security services has to be considered to avoid side channel attacks by monitoring of the exclusive locks, caused by the exclusive access to the services.

Line Out	X		Demand for prioritized exclusive access
Line In	Read only therefore no issue		
Graphical Out	X		Demand for prioritized exclusive access
Keyboard	X		Demand for prioritized exclusive access
Pointing Device	X		Demand for prioritized exclusive access
HMI and Audio Management		X	Prioritization has to be considered on the level of the management interfaces, see rows before.
CPU time	X		Sharing will be done by the virtualization subsystem, out of scope
Memory		X	Memory areas will be assigned to the partitions in a static manner by the virtualization subsystem and ensuring isolation, hence not an issue.

Table 31: Classification of resources concerning prioritized resource access

5.1.4 Special Issues concerning Prioritization in SVAS

Assuming that there is enough computational power to process any incoming and outgoing messages in the SVAS, the available data transfer rate of the underlying CAN bus is the theoretical limit. Nevertheless, CAN messages have different priorities, which have to be reflected in the whole output communication stack. Furthermore, if more than one partition is allowed to write a data object, there is an issue about prioritization on this level.

5.2 Concepts for Prioritized Resource Access in OVERSEE

Based on the two groups defined above, two concepts to describe the expected prioritized access to shared resources are used:

5.2.1 Resources shared over time

- Determine number of partitions, which are allowed to access this resource.
- Define a value for the priority of a partition concerning access to the current resource. The value should be between one (highest priority) and the number of partitions allowed to access this service. (identical values are allowed leading to a cyclic resource sharing between the partitions with the same priority value)

The priority information per resource and partition will be noted in a static way in an OVERSEE configuration file.

5.2.2 Resources with shared Capacity

- Perform the same steps as in section 5.2.1
- Add for each partition the minimum amount of capacity (e.g., data transmission rate) necessary to conduct the obligations, which are associated to this partition.

The implementation to follow this concept would assign the available capacity based on the priorities of the partitions for this resource. The main difference is that the implementation will only guarantee the defined minimum capacity. If a partition would need more capacity than defined, this capacity could only be provided if there is capacity left which was not requested within the minimum requirements by other partitions. See Figure 4 for a schematic view of an exemplary resource sharing.

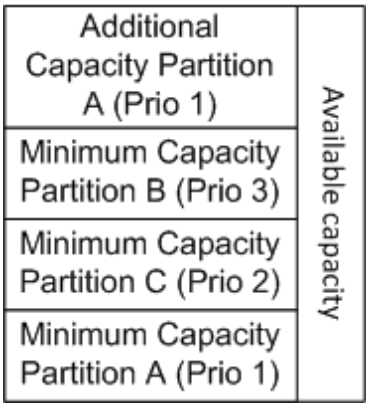

Figure 4: Example of prioritized capacity resource sharing

5.3 Change of Resource Sharing During Operation

If all partitions with high or highest priority for a resource would use this resource the whole time, obviously the whole concept does not make any sense. Therefore, it is important that the integrator, who is responsible for the allocation of the priority information, would be very careful, especially concerning the assignment of high priorities. Naturally, many applications (e.g., the depicted eCall application) would be in idle state most of the time, which leads to the current approach.

Concerning the resources with shared capacity it should be mentioned that the defined minimum capacity per partition is indeed not the current. This means that if a partition does not need its defined minimum capacity, this free capacity can be assigned to the other, maybe still requesting, partitions. These assignments would be for sure also based on the priorities of the other partitions. Nevertheless, the integrator should be again careful with the value for the minimum capacity assigned to each partition, especially in the cases of general-purpose partitions with probably malicious applications.

5.4 Policy Language for Definition of Prioritized Resource Access

To avoid an additional configuration file the firewall configuration file is extended in order to add the necessary configuration values for the prioritized resource access.

Extract from adapted OVERSEE firewall configuration file:

OVERSEE_firewal_policy_language.dtd, Beta, Version 0.2:

```
<!ELEMENT Partition EMPTY>
<!ATTLIST Partition
          name CDATA           #REQUIRED
          read (true|false)    #IMPLIED
          write (true|false)   #IMPLIED
          priority CDATA       #REQUIRED
          mincapacity CDATA    #IMPLIED
```

5.5 Security Challenges for Assurance of Prioritized Resources Access

As already showed by the eCall example there are security issues concerning the assurance of prioritized resource access. In the following, some of them are listed; this investigation is work in progress, which will be continued during the next steps of the project:

Denial of Service (DoS) attack

Entities could try to consume more resource capacity than have been assigned to them, or lock resources for exclusive access without having the highest priority of the requesting entities. If the attack is successful this could lead to situations where other entities are not able to fulfil their obligations due to missing resources.

Masquerading

Entities could try to masquerade themselves to use resources with the priorities assigned to other entities. This could be an attack vector for a successful DoS attack.

Forging of priority configuration

Entities could try to modify the configuration of the platform concerning prioritization; hence they will be able to request more resource capacity or a prioritized exclusive access to resources. Besides the value of the exceeding resource access this attack could also be a preparation for a DoS attack.

5.5.1 Residual Risks

Obviously, the limits of the proposed concept for the assurance of prioritized resources access are the borders of the platform, indeed the borders are much more closer, since for example some implementations of network interface drivers are out of scope of the generic OVERSEE implementation. Hence, DoS attacks on the external connections of OVERSEE cannot be avoided by OVERSEE security means.
Another residual risk is the question if the high level assurances of prioritized resource access could be preserved through the whole implementation (e.g., a communication path build out of different resources). This issue will be investigated within the rest of the project to discover possible additional security weaknesses.

6 Specification of internal communication requirements in the virtualized environment

As already mentioned above, integrated systems contain a number of software modules with different safety and security levels on a single hardware module. Therefore it is essential to partition these software modules in time and memory, so that errors and malicious users are contained in the module where they show up, and to prevent the propagation into other software modules (of even higher criticality), in order to guarantee the robustness of the resulting system.

Nevertheless many applications require the different software modules to communicate with each other. To guarantee a safe and secure communication channel, which does not allow erroneous/insecure software modules to influence the other software modules in the system, the interpartition communication system is used to monitor the communication and make sure that everything is all right. To prevent possible faulty software modules from directly or indirectly influencing other software modules, the interpartition communication system has to be designed properly.

6.1 High-Level Requirements

Some of the guiding criteria are:

- ideally only exclusive access at any point in time (no sharing of physical address ranges)
- Pre-defined communication end-points (no dynamic creation of communication channels)
- Bounded resources per channel
- All inter-partition synchronization happens at the application level (that is the core is lock-free)
- System monitoring can detect violation of pre-defined communication patterns
- Intervention allows to restrict violations of pre-defined communication patterns to a single partition

(from the OS perspective - it might well not be possible to continue operation properly with a communication partner having failed)

These high-level requirements are met best by re-using existing software concepts that were developed in the context of automotive and other HW and/or safety related systems (i.e. ARINC 653).

Since OVERSEE intends to implement the well specified and tested interpartition communication mechanisms described in ARINC653, the internal communication will be possible mainly via queuing ports and sampling ports. For some special cases where a high data throughput is necessary, the ARINC653 communication mechanisms will be complemented by a shared memory.

The specification of the first two is rather simple, since queuing ports and sampling ports as specified by the ARINC653 specification, and therefore these two are already very well specified.

Since shared memory is not specified in ARINC653, but is provided as a special feature for applications with a need for a communication mechanism with a high throughput, it has to be handled more rigorously.

The communication mechanisms presented by safety standards as ARINC653 build a strong basis for a secure communication between the OVERSEE partitions. For example, the static configuration of communication channels and ports restricts the access of unauthorized users/applications to the channels/ports intended by the system integrator. A proper design and use of these features should suffice to eliminate sniffing, man in the middle attacks, denial of service attacks and the like - so if one of the partitions should get hijacked, these mechanisms are the basis to contain the attacker and restrict his access to partitions communicating with the hijacked partition.

Furthermore the set of messages accepted by the receiving partition should be restricted to a well-defined set - this of course cannot be done within the OVERSEE project, but has to be left to the application developer, since messages are specific to the application.

It is the role of the integrator to provide the most restrictive system level configuration with respect to communication properties to ensure the minimum possible impact of a failing/compromised partition on others. In the context of OVERSEE only the mechanism is under investigation that allows to actually enforce such restrictive policies.

Although the communication between partitions as defined in ARINC653 follows a strict polling semantic, which also counteracts denial of service attacks, the traffic on the channels should be monitored and if partitions try to send more messages than the predefined threshold, actions should be taken (remove channel, discard messages, restart sending partition, etc.).

While these methods were not initially designed for security purposes it should be noted that the monitoring proposed by ARINC 653 well fit generally used methods in security related intrusion detection systems, thus there is a strong overlap of safety related mechanisms that can be utilized in the context of OVERSEE.

6.2 Available Communication Mechanisms

Before we get started on the requirements for the internal communication in the virtualized environment, the available communication mechanisms are summarized shortly.

6.2.1 Sampling Ports

Sampling ports can be used to send messages at any time, but they are restricted to fixed length messages. There exists only one copy of the message, which is overwritten every time a partition sends a new message. There is no buffering supported, therefore messages can be lost, but the data a reader gets, always contains the newest available instance of the message.

The resource management for sampling ports is pretty simple, since ARINC653 does not require any buffers at the sending end, and one buffer to hold the last valid value at the receiving end. Because the length of the message is known a priori (from the XML configuration file), the buffers can be set up at initialization time.

Messages, which are not of the right length are discarded, because ARINC653 [16] says in section 2.3.5.1:

"At the application level, messages are atomic entities i.e., either the whole message is received, or nothing is received. Applications are responsible for assuring, data meets requirements for processing by that application. This might include range checks, voting between different sources, or other means."

This paragraph precisely defines the border of jurisdiction between the application and the message passing system. The message passing system has to make sure that the message arrives in one piece at the receiver (which is definitely not possible, if the message has the wrong length), but it has no influence on the data itself, which is application specific, so the validity of the content cannot be of concern for the interpartition communication system.

Another possibility how a message on a Sampling Port could get invalid is aging. Each port gets assigned a REFRESH_RATE at configuration time. This rate defines the maximum age of the message, which is acceptable. If the message gets too old, it becomes invalid.

Sampling Ports support Uni-, Multi- and Broadcast. That means a message can be sent to one, multiple or all other nodes.

6.2.2 Queuing Ports

In contrast to Sampling Ports, Queuing Ports buffer the messages. The buffer shows FiFo behaviour and supports fixed as well as variable length messages. The buffering of the messages makes sure that no messages are lost, i.e. several instances of the same message are recognized by the receiving partition.

One important part of Queuing Ports is the buffer management. In contrast to sampling ports, here at least 2 buffers are needed. Both buffers are managed as FiFo message queues. One is dedicated to the sending side, and filled up with messages by the send requests of the sender. The second buffer is dedicated to the receiver. The messages are copied from the first buffer into the second by the interpartition communication system, and are removed from the buffer, after a receive request of the receiver has been served. The interpartition communication system takes care that only valid messages are copied into the second buffer, and the rest is discarded. New messages may be discarded, if the buffer is full, and an appropriate return code is given back to the sender.

In contrast to Sampling Ports, Queuing Ports support only Unicast Communication.

6.2.3 Compliance with RTE API

It should be noted that the semantics of queuing ports fits the semantic properties of standard high-level IPC mechanisms like POSIX message queues or sockets sufficiently well to allow a clean integration into high-level API models for the general purpose partitions. Further the mechanisms for communication used in OSEK/VDX can easily be mapped to the

communication primitives specified in ARINC 653. The mapping will require some syntactic glue but the semantic mapping is sufficiently aligned to expect reuse of existing software components at the runtime environment level. While this is not directly part of the OVERSEE effort we see this as an important design criterion as a mismatch at the semantic level would make it very hard (or impossible) to migrate existing applications to OVERSEE - which would make this effort futile.

6.2.4 Shared Memory

One of the problems of sampling and queuing ports is that they focus on classical control data - which is generally streaming by nature. OVERSEE can't impose such restrictions on the general automotive application domain - thus a mechanism for sparse data is mandatory. Traditionally shared memory has been used to communicate sparse data objects between concurrent processes.

Although a shared memory cannot be as robust as the sampling/queuing port mechanisms described above, the OVERSEE architecture offers a shared memory mechanism for applications that need a high throughput. An example would be data for the graphical interface.

The problem with a shared memory is, that the strict polling policy, which is followed by the sampling/queuing ports as specified by ARINC653 is not followed, introducing a dependency between the writing and reading applications.

Following a standardized shared memory API, as for example the POSIX shared memory API, would be a good choice.

6.2.5 Design Discussion - Robust SHM:

While this is not the place for a full design discussion we outline a preliminary concept for shared data.

6.2.5.1 Asymmetric Bi-directional mapping (ABM)

A possible resolution of this problem could be to include a bi-directional asymmetric mapping

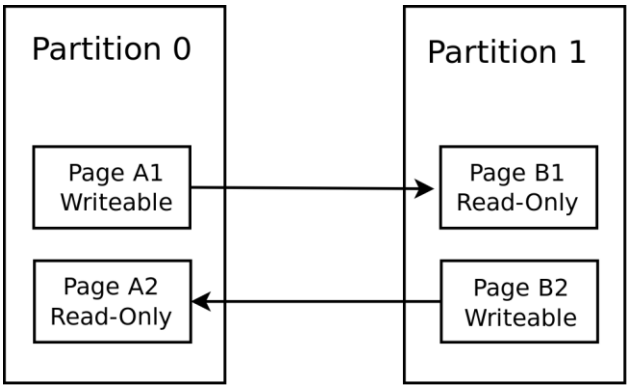

Figure 5: Asymmetric Bi-directional mapping (ABM)

Each of the shared pages is only writable for one partition - enforced by the underlying hardware memory management unit (MMU) and mapped into the Virtual Memory Area (VMA) of the respective communicating process. This way, while allowing to communicate sparse data objects, the security properties can be improved compared to fully write-shared pages.

6.2.5.2 Exchange Page Table Entry (XPTE)

A further mechanism could be to actually exchange page-mapping between communicating partitions by allowing the core OS to manipulate the respective page-tables. This way any content stays uniquely mapped to a single partition at any point in time while allowing to share sparse data.

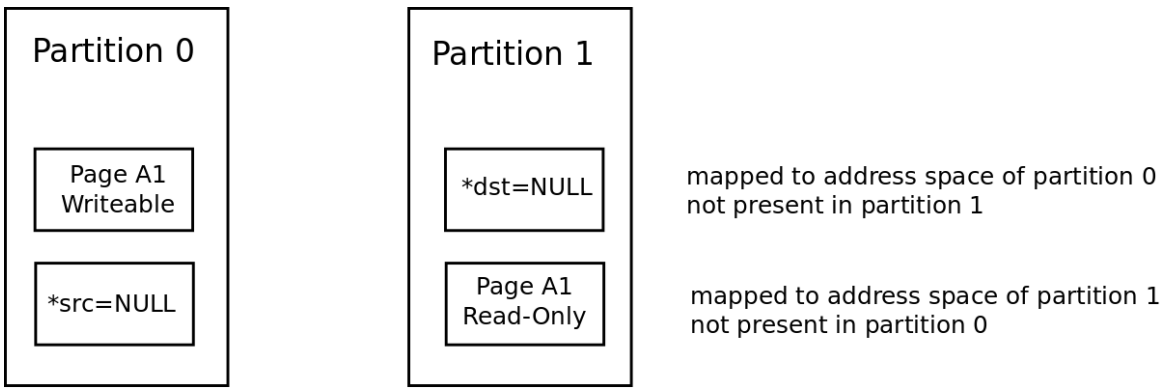

Figure 6: Exchange Page Table Entry (XPTE)

These mechanisms need further investigation - only early design efforts have been conducted to date.

6.3 Requirements on the Internal Communication

In order to analyse the requirements of the interpartition communication system, we developed 5 scenarios showing all possible and necessary ways of communication between partitions. By combining these models all access patterns can be constructed for secure and non-secure communication to/from applications running in partitions.

The figures in this section only depict ports, but these ports could also be shared memories.

6.3.1 Scenario1: Secure Communication - Dedicated Partition

This first scenario describes how a secure connection from an application to the outside world is made. The application partition is connected to the secure I/O partition via a virtualized communication channel; the validity of this channel is checked by the hypervisor's interpartition communication system. The secure I/O partition then applies its security services to the traffic it is routing between the application and the outside world. In this scenario, the driver for the peripheral is integrated into the secure I/O partition; the hypervisor provides virtualized interrupts of the peripheral to the secure I/O partition.

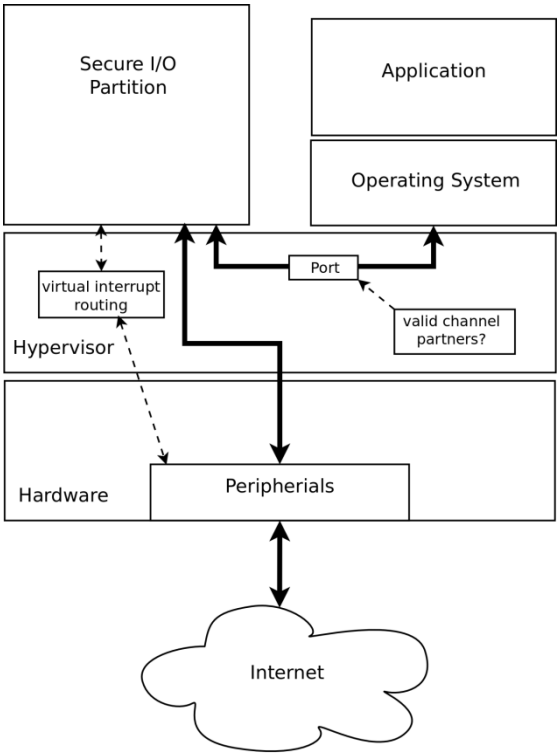

Figure 7: Secure Communication - Dedicated Partition

6.3.2 Scenario2: Non/low-secure device access - integrate in guest OS partition

The simplest scenario is an access to the outside world via an interface that is used by only one partition and where no or only low security measures are needed. In that case, the driver for the peripheral is integrated into the application partition, and all the hypervisor has to do is to route the virtualized interrupts for this peripheral to the application partition.

(Sharing can be achieved by the guest-OS acting as a server - while not secure this is the most flexible solution and suitable for non-security related services - i.e. browsing the web or entertainment related applications).

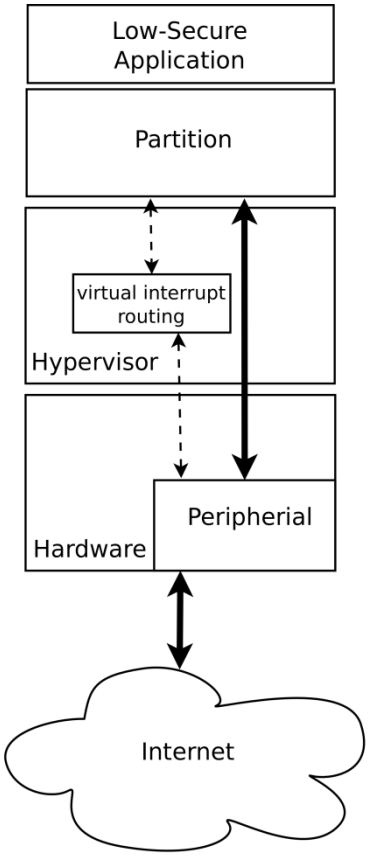

Figure 8: Non/low-secure device access - integrate in guest OS partition

6.3.3 Scenario3: Device Multiple Access - Virtual Multiplexing

Multiple applications can share a device via a channel multiplexer – the system partitions multiplexer need not be "intelligent" - it can simply duplicate all packets and present them to all partitions who then use what they see fit. Security is implemented at a different level in this model - if secure multiplexing were needed then the driver could be managed by a sufficiently "intelligent" partition that i.e. provides firewalling and routing services thus this is de-facto a virtual hub at the level of the VFB (in AUTOSAR taxonomy).

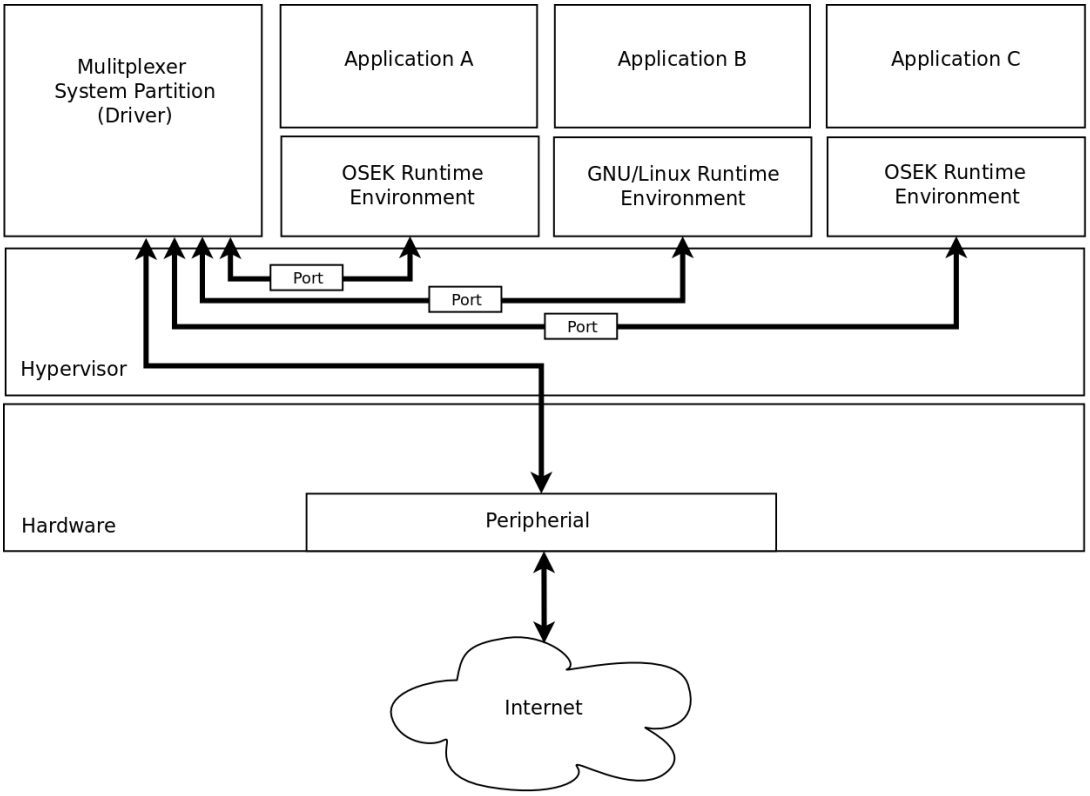

Figure 9: Device Multiple Access - Virtual Multiplexing

6.3.4 Scenario 4 - Virtual Device - Interpartition Communication

The big difference to Scenario3 is that the secure I/O partition does not just send the packets to all the partitions, but does a route so that ever partition gets only the traffic that is really intended to be for this partition. Furthermore, secure I/O partition does not only offer a connection to the outer world, but a secure channel between two application partitions. This scenario can be identified with a layer-3 router in the IP network sense.

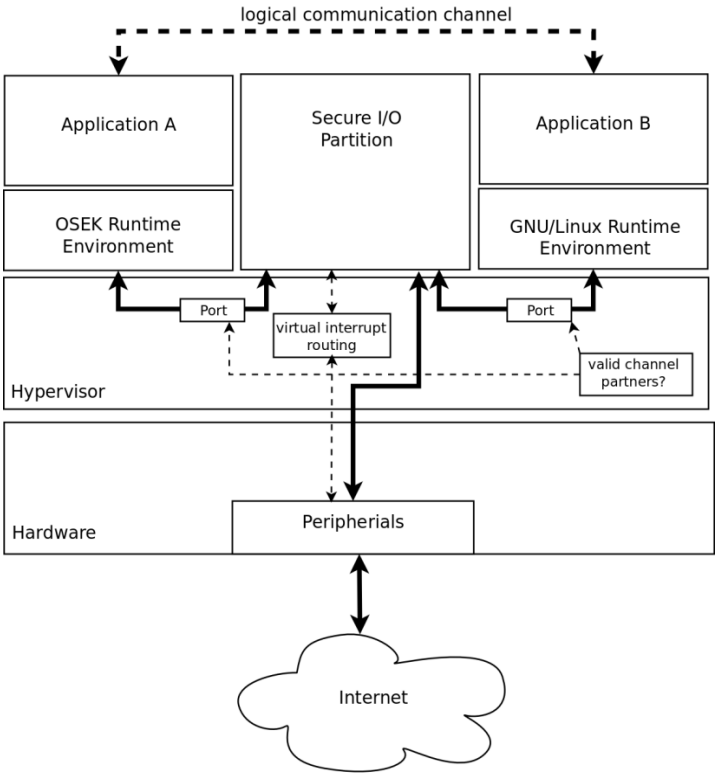

Figure 10: Virtual Device - Interpartition Communication

6.3.5 Scenario 5 - Scalability of the Architecture

Essentially, scenario 5 is not a basic communication model as the other 4, but already a combination of scenarios. This scenario shows how composable the OVERSEE architecture is, and how independent applications are made from the other applications on the platform. As the figure shows, there are two applications located on two separate hardware nodes, which need to communicate with each other. For the applications it does not matter, whether the other application is located on the same hardware node (Scenario 4), or on another one, it just connects to the secure I/O partition and sends its data. The secure I/O partition is configured by the integrator at compile time. Therefore it has a priori knowledge on the location of the applications. In case the receiver is on another hardware node, it just uses its pre-configured knowledge to send the data to this second node.

In this scenario the secure network service partition also has to maintain topology information or rely on external routing services (which could themselves have security implications). This scenario is currently not in scope for the on-going OVERSEE effort - though it is included to show that the generic nature of the driver abstraction model in principle allows to fully de-centralize computing capacity without, in principle, reducing security capabilities.

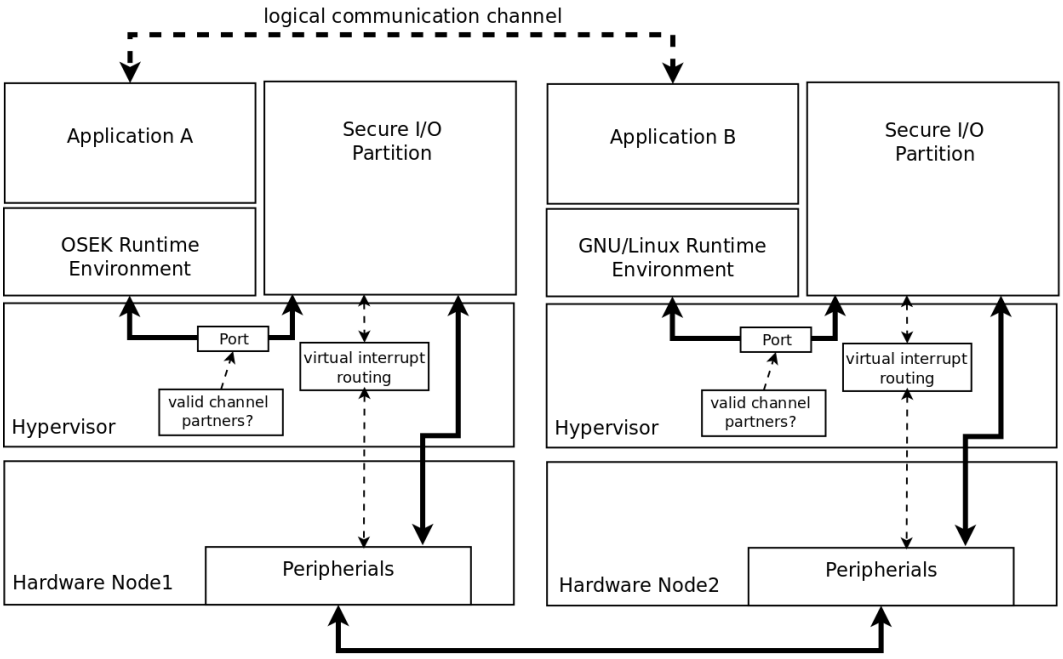

Figure 11: Scalability of the Architecture

7 Next Steps

Since with the finalization of this deliverable and the other related deliverables in WP2 as well as the D3.1 "Selection for reuse of existing building blocks" the overall design for OVERSEE is available; the second milestone of the project has been reached.

Next the project will enter a new phase, the OVERSEE implementation phase, which is mainly corresponding to WP3 and WP4.

Anyway, since it is quite reasonable that during the implementation phase of the project new aspects and tasks will arise, it would be probably necessary to go back into the design phase for some aspects. Hence, we would follow an iterative process whenever necessary.

References

[1] OVERSEE Project: D1.1 Use Case Identification. 2010

[2] OVERSEE Project: D1.4 Functional Requirement Analysis. 2010

[3] OVERSEE Project: D2.2 Specification of security services incl. virtualization and firewall mechanisms. 2011

[4] OVERSEE Project: D2.4 Specification of secure communication. 2011

[5] XtratuM, www.xtratum.org/

[6] OSEK/VDX: www.osek-vdx.org

[7] OSEK Operating System Specification 2.2.3: portal.osek-vdx.org/files/pdf/specs/os223.pdf

[8] OSEK/VDX Operating System Test Plan: portal.osek-vdx.org/files/pdf/modistarc/ostestplan20.pdf

[9] ETSI: TS 102 637-1 V1.1.1. Intelligent Transport Systems (ITS); Vehicular Communications; Basic Set of Applications; Part 1: Functional Requirements. Sep. 2010

[10] ETSI: TR 102 893 V1.1.1. Intelligent Transport Systems (ITS); Security; Threat, Vulnerability and Risk Analysis (TVRA). Mar. 2010

[11] ETSI: TR 102 638 V1.1.1. Intelligent Transport Systems (ITS); Vehicular Communications; Basic Set of Applications; Definitions. Jun. 2009

[12] ETSI: EN 302 665 V1.1.1. Intelligent Transport Systems (ITS); Communications Architecture

[13] ETSI: ES 202 663 V1.1.0. Intelligent Transport Systems (ITS); European profile standard for the physical and medium access control layer of Intelligent Transport Systems operating in the 5 GHz frequency band

[14] eSecurity Working Group: Vulnerabilities in Electronics and Communications in Road Transport: Discussion and Recommendations. Jun. 2010

[15] National Marine Electronics Association: NMEA 0183 Standard. www.nmea.org

[16] ARINC: Avionics Application Software Standard Interface

[17] M. Masmano, I. Ripoll, A. Crespo, V. Brocal: XtratuM Hypervisor for LEON2 Volume2: User Manual. Sep. 2009

[18] OVERSEE Project: D2.1 List of interfaces and specifications of information flow. 2010

[19] W3C: Extensible Markup Language (XML). www.w3.org/TR/xml/

[20] eSafety Forum, "Recommendations of the DG eCall for the introduction of the pan-European eCall", Apr. 2006, Version 2.0

[21] NIST Special Publication 800-38E: Recommendation for Block Cipher Modes of Operation: The XTS-AES Mode for Confidentiality on Storage Devices

[22] J. Rushby. Design and verification of secure systems. volume 15, pages 12–21, Pacific Grove, California, Dec. 1981

[23] R. Goldberg. Survey of virtual machine research. IEEE Computer Magazine, 7(6):34–45, 1974