Project acronym:	EVITA
Project title:	E-safety vehicle intrusion protected applications
Project reference:	224275
Program:	Seventh Research Framework Program (2007–2013) of the European Community
Objective:	ICT-2007.6.2: ICT for cooperative systems
Contract type:	Collaborative project
Start date of project:	1 July 2008
Duration:	36 months

Deliverable D3.3: Secure On-Board Protocols Specification

Authors: Hendrik Schweppe, Sabir Idrees, Yves Roudier (EURECOM); Benjamin Weyl (BMW Group Research and Technology GmbH); Rachid El Khayari, Olaf Henniger, Dirk Scheuermann (Fraunhofer Institute SIT); Gabriel Pedroza, Ludovic Apvrille (Institut Télécom); Hervé Seudié (Robert Bosch GmbH); Hagen Platzdasch (Infineon Technologies AG); Michel Sall (Trialog)

Reviewers: Anselm Keil (Continental Teves AG & Co. oHG); Marko Wolf (escrypt GmbH)

Dissemination level:	Public
Deliverable type:	Report
Version:	1.4
Submission date:	15 July 2011

Abstract

The objective of the EVITA project is to design, verify, and prototype an architecture for automotive on-board networks where security-relevant components are protected against tampering, and sensitive data are protected against compromise. Thus, EVITA will provide a basis for the secure deployment of electronic safety aids based on vehicle-to-vehicle and vehicle-to-infrastructure communication. This document specifies security protocols for on-board communication. These security protocols work on different layers of the protocol stack and include keying protocols using interactions with hardware security modules. We show that it can be assured (a subset of the secure on-board protocols presented are formally verified), that security requirements such as confidentiality, authenticity and freshness of communication channels are met.

Contents

List of figures

List of tables

List of abbreviations

3DES	Triple DES
3GPP	3rd Generation Partnership Project
AES	Advanced Encryption Standard
API	Application Programming Interface
AUTOSAR	Automotive Open System Architecture
CA	Certification Authority
CAM	Cooperative Awareness Message
CAN	Controller Area Network
CCM	Communication Control Module
CCU	Communication Control Unit
CLI	Caller Line Identification
CMAC	Cipher-Based MAC
CPU	Central Processing Unit
CRC	Cyclic Redundancy Code
CRS	Cryptographic Services
CSMA	Carrier Sense Multiple Access
CSMA/CA	Carrier Sense Multiple Access/Collision Avoidance
CSMA/CD	Carrier Sense Multiple Access/Collision Detection
CTP	Common Transport Protocol
DES	Data Encryption Standard
DSA	Digital Signature Algorithm
DSAP	Delegate-Specific Authorization Protocol
DSRC	Dedicated Short-Range Communications
DT	Diagnostic Tool
ECC	Elliptic Curve Cryptography
ECR	ECU Configuration Register
ECU	Electronic Control Unit
EEPROM	Electrically Erasable Programmable Read-Only Memory
GPRS	General Packet Radio Service
GPS	Global Positioning System
GSM	Global System for Mobile Communications
HIS	Hersteller-Initiative Software
HMAC	Hash-Based MAC
HMI	Human-Machine Interface
HSM	Hardware Security Module
HU	Head Unit
IDK	Device Identity Key
ILP	Inter-Layer Proxy
IP	Internet Protocol
IV	Initialization Vector
KM	Key Master
LIN	Local Interconnect Network
MAC	Message Authentication Code
MAC	Media Access Control

MNO	Mobile Network Operator
MOST	Multimedia Oriented Systems Transport
MVK	Manufacturer Verification Key
NIST	National Institute of Standards and Technology
N-PCI	Network Layer Protocol Information
NTP	Network Time Protocol
OEM	Original Equipment Manufacturer
OIAP	Object-Independent Authorization Protocol
OSAP	Object-Specific Authorization Protocol
PCR	Platform Configuration Register
PDM	Policy Decision Module
PDU	Protocol Data Unit
PEP	Policy Enforcement Point
PIN	Personal Identification Number
PKI	Public-Key Infrastructure
PSAP	Public Service Answering Point
PSK	Pre-Shared Key
RAM	Random-Access Memory
RFU	Reserved for Future Use
RSA	Rivest, Shamir, and Adleman Public Key Encryption Technology
RSU	Road-Side Unit
RTC	Real Time Clock
SAML	Session Assertion Markup Language
SCI	Serial Communications Interface
SeBB	Security Building Block
SFOTA	Secure Firmware Update over the Air
SRK	Storage Root Key
SSL	Secure Sockets Layer
SSM	Secure Storage Module
SWD	Security Watchdog Module
TCG	Trusted Computing Group
TCP	Transmission Control Protocol
TLS	Transport Layer Security
TPM	Trusted Platform Module
UART	Universal Asynchronous Receiver Transmitter
UDP	User Datagram Protocol
UML	Unified Modeling Language
UTC	Coordinated Universal Time
V2I	Vehicle to Infrastructure
V2V	Vehicle to Vehicle
V2X	Vehicle to Vehicle (V2V) and/or Vehicle to Infrastructure (V2I)
VANET	Vehicular Ad-Hoc Network
VM	Virtual Machine
XML	Extensible Markup Language

Document history

Version	Date	Changes
1.0	2010-07-20	Version 1.0 submitted
1.1	2010-09-15	Clarification of key distribution and export flags; inclusion of time synchronization protocol and secure storage protocol
1.2	2010-12-21	Updated notations of secure transport protocol, revised firmware update protocol.
1.3	2011-02-01	Added details for integration of SeVeCom communication stack in Appendix A.1.
1.4	2011-07-15	Integrated feedback from T3400, protocol verification.

1 Introduction

This section provides background information, the purpose and scope of the secure on-board protocols and an outline of this deliverable.

1.1 Background

The automotive on-board network is a heterogeneous network of Electronic Control Units (ECUs), sensors, actuators and communication bus systems. Communication between individual ECUs and between ECUs and sensors and actuators is necessary for real-time control loops. In today's vehicles there exist a number of different bus systems, of which the Controller Area Network (CAN) bus is the most prominent one. While it cannot guarantee a minimum latency, its efficiency lies in a Carrier Sense Multiple Access/Collision Avoidance (CSMA/CA) arbitration scheme where message IDs are used for prioritizing messages on the bus and collisions do not occur by design. Existing vehicular bus systems are not designed to offer protection against malicious attacks on the communication. For dedicated use cases, respective security protocols are employed, such as authentication for flashing ECUs or authorization of certain software capabilities. Bus systems are not physically accessible (apart from the diagnosis interface); however, they only partly provide logical means of security.

With the increasing interconnection of the vehicle with external communication entities, such as mobile devices, internet and infrastructure services, other vehicles, Road-Side Units (RSUs), etc. new threats need to be considered in order to secure the automotive on-board network against malicious attacks [55]. Specifically, in order to secure e-safety application scenarios, dedicated security protocols need to be designed, e.g., in order to identify and filter manipulated and fake messages. The design of security protocols is based on the security architecture described in [64].

1.2 Purpose and Scope

The scope of this specification is the secure communication between entities within the automotive on-board network, i.e. between ECUs, sensors and actuators, as well as controlled access to functions and data. Providing security for uni-, bi- and multidirectional intra-vehicle communication with regard to authenticity and other security properties, as needed, is the main target of this work. External secure communications, e.g. within vehicle-to-vehicle communication scenarios as described in various other research activities, such as the SeVeCom [37] project, are out of scope except when they are relevant for managing the security of the on-board security system. Respective security interfaces between security components for secure internal and external communication are specified, where appropriate.

The specification of secure on-board protocols is done considering the requirements of future safety use cases. The security protocols are designed as plug-ins of the security software architecture described in [64]. These security protocols are able to securely connect EVITA nodes (i.e., ECUs, sensors and actuators) of different underlying network technologies (ranging from low-speed CAN to high-speed Ethernet). The separation paradigm of the security software framework is taken up so that applications and high-level protocols are agnostic of lower level networking specifics. Where appropriate, other projects and standards are referenced within this document.

Subject Matter of Secure On-Board Protocols Specification This specification comprises:

- the security protocols themselves for establishing security properties among entities,
- the security management protocols for configuring and maintaining the security system,
- the policies, e.g. access, firewall, or intrusion detection policies,
- the instantiation of a set of protocols and policies with the deployment within a dedicated on-board network architecture.

Key Design Objectives The secure on-board protocols shall meet the desired security requirements of a set of applications defined in D2.1 [33]. The following key objectives are followed during the design of the security protocols:

- Efficient security protocols applicable within embedded automotive environments,
- Scalable and flexible security protocols that can be configured according to the specific security requirements of a set of applications,
- Modularization in such a way that security protocols can be combined and, when appropriate, interchanged while preventing that certain security functionality is implemented redundantly,
- Consideration of vehicular off-line as well as on-line scenarios in order to manage the on-board security system.

Design Goals In order to achieve these key objectives, EVITA has the following key design goals for the security protocol specification:

- to provide protocols and interfaces which can be tuned according to a set of preconditions and constraints of a specific on-board architecture,
- to provide configurable protocol interfaces in order to configure the security and trust level according to the risk analysis and security requirements,
- to specify inter-plugable protocols, i.e., a protocol can work stand-alone and can be extended at certain interfaces with additional protocols,
- to design security protocols that can be deployed in centralized, multi-centered and completely distributed environments,
- to design mechanisms for managing the security protocols, keys and policies in on-line and off-line scenarios,
- to define policies required for specific use cases.

1.3 Contributions of the Document

In this deliverable, we show how our security architecture, which includes a powerful software security framework as well as Hardware Security Modules (HSMs), can be used for communication in the vehicle. We propose to use an additional anchor of trust in the vehicle to secure communication between low-level sensor nodes and high-level processing units. For the protocols proposed in this deliverable, we show how modules of the security framework are used accordingly (for both hardware and software). We give an example for deployment of security modules with regard to protocols and evaluate the approach in a dedicated simulation section, which gives already an outlook on the security validation in [16].

1.4 Outline of the Document

Section 2 gives an overview of related protocols for automotive on-board communication. The major part of these protocols is not concerned with security. We enhance them with security in Section 3. Section 4 describes how EVITA protocols can be modeled using TTool [42]. Section 5 contains concluding remarks. Appendix A provides technical details of related security protocols and of automotive applications, such as diagnosis, automated emergency calls, or automated road tolling, and explains how these work.

2 Related Work

During the work on on-board protocols, we have researched a large amount of related work. We cover many different domains and use cases with our protocols, and base these on existing work where possible and sensible. Many details about these related protocols can be found in Annex A, while we cover the most important topics in this section. We give an outlook on the constraints of today's bus systems in the following and subsequently discuss authentication and related security protocols.

2.1 Protocols for automotive on-board communication

The electronic components of an automotive on-board network are connected via various communication bus systems such as CAN, Local Interconnect Network (LIN), FlexRay, or Multimedia Oriented Systems Transport (MOST) using the corresponding transport protocols such as those described in [29]. Additional information about transport protocols for automotive network can be found in [65]. These bus systems are not primarily concerned with security, but provide services upon which higher-layer security protocols depend. The following subsections briefly introduce these bus systems in terms of

- packet or circuit switching,
- addressing scheme,
- data rate,
- transmission guarantee, and
- maximum block size.

The bus systems are typically designed for the physical and data link layer. Transport protocols such as ISO-TP for CAN [29] and AUTOSAR-TP [9] for FlexRay are used at the transport layer. Also Internet Protocol (IP) may be used in on-board networks: This is currently analyzed in the German project SEIS [59].

CAN The CAN protocol described in [20] has been specified and developed by Robert Bosch GmbH as communication protocol for vehicle on-board communication. CAN specifies a bus system, where all nodes have the same rights. The sending of messages by nodes is prioritized through the use of CAN IDs. Those IDs are part of every message. The message with the smallest CAN ID value has the highest priority. Every message is sent to the CAN bus (broadcasting) and each node decides on the basis of the ID whether the message is relevant for him. Therefore, the access to the CAN bus is done through Carrier Sense Multiple Access (CSMA). Each node monitors the bus and sends a message only if it is required and the bus is free.

Some characteristics of the CAN protocol are:

- Packet switching,
- Addressing scheme: Use of CAN identifiers, which typically have a length of 11 bits,

- Data rate: Max 1 Mbit/s,
- Transmission guarantee: High,
- Maximum block size: 8 Bytes.

Additional information on CAN can be found in the following CAN standards [21, 22, 23, 24, 25]. There are a lot of transport protocols for CAN. The most popular is ISO-TP standardized in [29].

FlexRay The FlexRay protocol is specified by the FlexRay consortium [10]. FlexRay provides flexible data communications by supporting three different topologies (bus, star and mixed) and a higher data rate than other protocols such as CAN. FlexRay allows both asynchronous transfer and real-time data transfer and operates as dual-channel system where each channel delivers a maximum data bit rate of 10 Mbps. FlexRay is a time-triggered protocol. It uses a continuous communication of all connected nodes via redundant data buses at predefined intervals. Each ECU has a given time slot to communicate. The slots are allocated at design time. A masterless protocol is used for synchronizing the ECU clocks by measuring time differences of arriving frames.

Some characteristics of the FlexRay protocol are:

- Packet switching,
- Addressing scheme: 11 bit address,
- Transmission guarantee: High,
- Data Rate: 2 x 10 Mbit/s,
- Maximum block size: 254 bytes.

Additional information on the FlexRay protocol as well as an interactive introduction can be found in [17]. The transport protocol AUTOSAR TP used for FlexRay is specified by the AUTOSAR Consortium [9].

LIN The LIN protocol is specified by the LIN Consortium [11]. LIN has been designed with the goal to have a low cost bus system for simple sensor-actuator applications. LIN is today a de-facto standard for the applications where the bandwidth and the versatility of CAN is not needed. Like FlexRay LIN is a time-triggered protocol. LIN is single-wired and based on the common Serial Communications Interface (SCI) (Universal Asynchronous Receiver Transmitter (UART)) byte-word interface. The communication is controlled by a master ECU that starts every communication. The master ECU sends periodically (in defined time slots) messages to slave ECUs. Exactly one of the slave ECUs reacts with an answer to the message of the master ECU. The master ECU uses schedule tables which are statically configured. The master ECU is often used as gateway to other buses. The communication partners in a LIN bus system are typically under 12 nodes. There are 64 identifiers. A particular feature of LIN is the synchronization mechanism, which allows the clock recovery by slave ECUs without quartz or ceramics resonator.

Some characteristics of the LIN protocol are:

- Packet switching,
- Addressing scheme: IDs of 8 bit length – 6 bit for the actual ID and 2 bit for parity,
- Data Rate: 20 Kbit/s,
- Transmission guarantee: High,
- Maximum block size: 8 bytes.

Additional information on the LIN Protocol can be found in [11].

MOST The MOST protocol specified by the MOST cooperation [12] is the de-facto standard for multimedia and infotainment networking in the automotive industry. The MOST bus is media oriented and designed to transport data streams. MOST often employs a ring topology and can include up to 64 nodes. In every block of 16 frames, one control-frame is transmitted on the control channel. Source and destination addresses of 16 bits each are used. Addresses are obtained from the master node and must be unique on one bus. Payload of control-channel frames consist of up to 17 bytes for MOST25 and up to 21 bytes for MOST50 (for which the control frame is distributed over all 16 frames of a block). The new standard MOST150 from 2007 [12] allows to transmit regular Ethernet frames encapsulated in asynchronous MOST messages. The MOST bus uses expensive transceiver units and is thus only used where an increased bandwidth is needed.

Some characteristics of the MOST protocol are:

- Packet switching,
- Addressing scheme: 16 bit functional address,
- Data Rate: Max 150 Mbit/s,
- Transmission guarantee: High,
- Maximum block size: 1024 bytes.

MOST is the only one of the automotive bus systems providing a specification of all seven ISO/OSI protocol layers. Additional information on MOST can be found in [12].

Table 1 presents a summary of the different bus systems.

Bus Systems	Bandwidth	Topology	Address Space (bit)	MTU (byte)	exp. Latency	Ordered
CAN	max. 1 Mbit/s	Bus	11	8	260 μs	no
LIN	20 kbit/s	Bus	8[1]	8	10 ms	no
MOST	max. 150 Mbit/s	Ring	16	1024[2]		yes
FlexRay	2× 10 Mbit/s	Bus, star, mixed	11	max. 254	35 μs[3]	yes

Table 1 Comparison of automotive network protocols

2.2 Authentication Protocols

For automotive on-board networks, authentication protocols are needed for two different general purposes: First of all, ECUs need to prove their identities against each other (entity authentication). Furthermore, messages transmitted within the on-board network need to prove their origin, i.e. from which entity they have been sent (data origin authentication). This basically results from the EVITA requirements specified in [55]

2.2.1 Entity Authentication

The authentication of ECUs within the on-board network is performed by so called "challenge-response methods" involving the exchange of recently generated random numbers (see e.g. [56] for closer descriptions). Each ECU owns appropriate secret or private keys and proves its identity by applying such a key to a random number generated by the intended verifying ECU. The identity is then verified by checking whether the key has really been applied to the previously issued random number. By this way, the authentication procedure basically consists in the combination of cryptographic algorithms with the issue and later verification of actual random numbers.

There are various possibilities for the design of entity authentication protocols to be realized for on-board networks. First of all, the authentication may be performed with symmetric or asymmetric cryptographic methods. In case of symmetric methods, claiming and verifying ECU must share a common secret key. The cryptographic operations may then consist either in an encryption and the decryption of a random number or the application of a keyed hash function (Message Authentication Code (MAC)) by the claiming ECU the result of which then needs to be verified. In case of asymmetric methods, the claiming ECU owns the private key which is then used for encryption or signing the random number. The verifying ECU then applies the corresponding public key for decryption or signature verification.

Depending on the intention, entity authentication protocols can be used for unilateral authentication, i.e. only one of the entities authenticates against the other or for mutual authentication, where two entities authenticate against each other. The choice between unilateral and mutual authentication is made based on the security requirements. For entity authentication protocols based on symmetric encryption, the amount of data to be exchanged only depends on the size of the random numbers and other identity data to be included. In case of keyed hash functions and signatures, it additionally depends on the size of MAC or signature. The number of messages to be exchanged for unilateral or mutual authentication is the same for all cryptographic methods.

Zero-knowledge authentication protocols (where the data to be transferred definitively do not contain any secret information) are even more secure than challenge-response protocols. However, they are impractical for on-board networks due to the large number of messages to be transferred which would cause a high amount of latency.

For more details on entity authentication protocols, see Appendix A.6.1.

[1]6 bit for the message ID. 2 bit for the parity
[2]1 Block = 16 frames. 1 Frame = 512 bit
[3]For 2 byte payload

2.2.2 Data Origin Authentication

Authentication of the origin of messages inside the on-board network is performed by the application of digital signatures (based on public key cryptography) or MACs (based on symmetric keys) as authentication data to the message data. MACs may be based either on block cipher methods or on keyed hash functions. Beside the origin of the message, this type of authentication data also proves that the message has not been altered during transmission. For avoiding replay attacks, additional information like time information may be included.

The claiming ECU calculates the authentication data and transmits it together with the message data. The verifying ECU then checks the correctness of the authentication data, i.e. whether it really corresponds to the message data.

The amount of data to be transmitted for the purpose of data origin authentication with digital signatures or MACs only depends on the length of the authentication data, i.e. the algorithm parameters used for the generation.

The data origin authentication protocol does not need any transfer of additional messages within the on-board network. It just consists of a modification of the original data message to be transferred.

For more details on data origin authentication, see Appendix A.6.2.

2.3 Authorization Protocols

Access control is a central issue in many systems and applications where multiple parties are represented, either as users or as stakeholders. Access control is entirely about describing resources, who can access them, and controlling that access. As explained in [7], *"Authorization is the means of determining whether a user, application or process is permitted to perform a particular operation"*. For automotive on-board networks, flexible and distributed on-/off-board user/identity authentication and authorization (access control) are needed in order to satisfy the needs of a diverse set of applications. For the realization of the distributed authorization mechanisms several protocols are already available and can be used.

Session Assertion Markup Language (SAML) is one of the protocol candidates. The SAML is an Extensible Markup Language (XML) standard for assertions regarding identity, attributes and entitlements of a subject [41]. It allows to exchange authentication and authorization data between security domains, that is, between an identity provider (a producer of assertions) and a service provider (a consumer of assertions). The service provider relies on a SAML assertion from the identity provider about the principal to make an access control decision. This setup requires the existence of local authorization services from an identity provider, however, SAML provides a level of abstraction since it does not specify how they are implemented.

Generally, a SAML assertion is a statement made at a given time by an issuer regarding a subject provided that certain conditions hold. SAML assertions can contain three types of statements: authentication, attribute and authorization decision statements. Each of these corresponds to a type of query which forms part of a SAML request-response protocol. The structure of an assertion is described by a SAML profile, which may be defined dependent on the desired application. At the implementation level, SAML messages admit bindings to several standard message types and protocols [41]. SAML provides XML formats for transmitting security information, defines how they work with underlying protocols, and specifies message exchanges for common use cases. In addition, it supports several privacy protection mechanisms (providing means to determine security attributes without revealing identity), and specifies a schema that allows systems to communicate the SAML options they support.

Trusted Computing Group (TCG) Authorization Protocols The specification v1.2 of the Trusted Platform Module (TPM) defines the main functions of a TPM, including its architecture, the management of its keys, credentials and authorization protocols. The purpose of the authorization protocols and mechanisms is to prove to the TPM that the requester has permission to perform a function and use some object. There are three protocols to securely pass a proof of knowledge of authorization data from requester to the TPM.

1. Object-Independent Authorization Protocol (OIAP). The OIAP supports multiple authorization sessions for arbitrary entities. The OIAP sessions were designed for efficiency, and only one setup process is required for potentially many authorizations of TPM entities. It authenticates the authorization data of each entity and protects the messages integrity. But it cannot encrypt the communicating messages.

2. Object-Specific Authorization Protocol (OSAP). The OSAP supports an authentication session for a single entity and enables the confidential transmission of new authorization information. In OSAP, only one setup process is required for many authorizations, and the entity authorization data is required only once. The OSAP creates a temporal secret that is used throughout the session instead of the authorization data. The temporal secret can be used to provide confidentiality and integrity of the following messages during the session. An OSAP session is efficient because only one session is required to authenticate the authorization data of many entities.

3. Delegate-Specific Authorization Protocol (DSAP). The DSAP supports the delegation of owner or entity authorization. The DSAP session is to provide delegated authorization information when a requester wants to access some resources in the TPM. Most parts of DSAP are the same as OSAP, and the main difference between them is the delegation privileges and the authorization data are needed when a user sends a request to set up a DSAP session.

2.4 Secure flashing protocols

Mahmud et al. [44] present a security architecture and discuss secure firmware upload. There are, however, a number of prerequisites and assumptions (i.e., sending multiple copies to ensure firmware updates) in order to make secure firmware update. However, sending multiple copies is not realistic and imposes several constraints on the infrastructure. This proposal does not consider automotive on-board networks, where domains are traditionally separated, due to functional and non-functional requirements. Furthermore, on-board key management issues are not mentioned in their approach. Kim et al. [34]

present remote progressive updates for flash-based networked embedded systems. In their proposed solution a link-time technique is proposed that reduces the energy consumption during firmware installation. However, no security aspects are mentioned in this proposal.

Nilsson et al. in [50, 51] provide a lightweight protocol and verification for Secure Firmware Update over the Air (SFOTA) in vehicles. In the SFOTA protocol, different properties are ensured during firmware update protocol (i.e., data integrity, data confidentiality, and data freshness). However, this approach also imposed several assumptions in order to ensure the secure software upload such as an authentication of the vehicle is not considered, keys are stored securely and single encryption key for each car. Furthermore, no specific execution platform requirements are put forward by this proposals. For instance, switching the ECU into reprogramming mode.

In [49], key management issues are discussed while performing software updates. A rekeying protocol is defined in order to distribute keys with only specific nodes in the group. It also use the multicast approach to update the software on a group of node. However, we consider that different firmware are installed on different ECUs, depending on the ECU functionalities, which makes multicast approach not useful. Furthermore, as mentioned above, this approach also does not consider execution platform requirements. It does not discuss about computation attacks, where, attacker can learn and modify the firmware, during installation phase or simply prevent to update the counter, for later replay attacks.

The most related approach with our work is Hersteller-Initiative Software (HIS) [46]. The flashing process provided by HIS provides a good basis for the Original Equipment Manufacturers (OEMs), but the recommended protocols do not provide all necessary security functionalities (e.g., freshness). Furthermore, this process only considers hard-wired firmware updates and does not provide any information about which key is used for firmware encryption in a very heterogeneous landscape of communication network technologies.

2.5 Embedded security

There exist many domains in the embedded world, in which on-board security is essential. The field of transportation systems (covering avionics, ships, and railway systems besides road vehicles) is closely related. However, there the security has been driven by safety requirements in the last decades and open documentation is not available to the public. With the increasing demand for consumer electronics, a different way has been taken for the latest aeroplanes, where entertainment bus systems are loosely coupled to vital on-board networks. In road vehicles, domains are traditionally separated. While this separation is advantageous for security requirements, it originated in functional requirements such as bus speeds and bus load as well as a minimized delay between frequently communicating ECUs.

With research and development going on on equipping cars with wireless communication interfaces, also work on securing the automotive on-board networks against malicious attacks has been started and there exist several approaches [6, 58, 18]. While [58] presents an approach that takes hardware into account by providing a secured runtime environment with a so-called Trust Zone on an ARM processor, the solutions of [6, 18] are software based. The so called *tools* and *enablers*, which are low-level and application-level security functions in [6] also cover a number of on-board automotive use-cases, while leaving the essential link to the external communication domain uncovered. The approach [18] is made up of a virtualized runtime environment for using AUTOSAR and multimedia functionality on the same ECU. Security functionality is build into the operating system core and does not go beyond the controller's bounds.

As can be seen e.g. in [6], the approach to separate security functionality into small blocks and consequently use these to build system-wide security, which we also follow, is common practice:

- **Low Level Security**, which provides core security functionalities that are specifically tailored for the application in vehicular domains. These cover cryptographic algorithms and methods as well as access control primitives for vehicular systems.
- **High-Level Security**, serves as major building block for applications and system security. This covers higher-level cryptographic handling (e.g., of certifications and for authorization and authentication tickets). The high-level security functionality relies on lower level services and realizes security applications such as code signing or secure update processes.
- **Application Level Security**, is provided as Application Programming Interfaces (APIs) to be used by third parties in order to implement custom security functionality and to interact with on-board authentication and authorization resources.

The EVITA approach, which includes an in-depth risk and attack analysis as well as secure hardware and a comprehensive security software framework, extends the related work found today on this topic.

3 Protocol Design

Using the security engineering process of [32], we specify a set of security protocols protecting the on-board network with respect to dedicated use cases. These protocols provide the link between the security requirements [55] inferred from the use cases [33] and the security architecture [64]. The use-case scenarios described in [33] define the assets that need to be protected against threats identified in the security and risk analysis [55]. Depending on the security objectives targeted for a certain set of applications, the on-board system needs to provide appropriate security services which are defined by the overall security architecture comprising the security protocols.

The security architecture and its HSMs defined in [64] provide the foundation for the security services. Each security protocol defined in the following is part of this architecture and is integrated with this architecture as so called plug-in (as defined in [64]). The security architecture modules need to communicate for remote deployment scenarios as well as locally. We take design constraints of the vehicle into account, while providing enhanced solutions for future vehicle systems with less constraints.

3.1 Introduction to EVITA security protocols

3.1.1 Use-case groups

To facilitate the selection of security protocols, we have grouped the use cases of [33] in order to identify similar communication patterns amongst the use cases. As shown in Figure 1, we have identified three major groups (clusters) of use cases that follow similar communication patterns:

- external communication (such as V2V and/or V2I (V2X) communication),
- maintenance communication (diagnosis and flashing), and
- integration of mobile electronics such as third party devices.

Individual use cases of one cluster may demand different security levels (e.g., Active Brake and Local Danger Warning share a similar message distribution, but have different security requirements), so that these have to be addressed individually. In later subsections, we focus on providing appropriate security features to protocols deployed in these use-case groups. Protocols are designed with flexibility in mind, so that different security levels can be used for communication by adapting the protocols.

Some use cases do not match any of the above clusters. The integration of the ungrouped use cases and their respective protocols is discussed separately. In the Appendices A.2 and A.3 protocols for eCall and eTolling are described. These are currently in the standardization process or under discussion for standardization. We take these up later in this section and describe how interfaces can be provided to enable these use cases.

3.1.2 Security Protocols addressed by EVITA

This section provides an overview of the security protocols specified in the EVITA project. For each use case cluster (cf. Section 3.1.1), a certain set of security protocols is required.

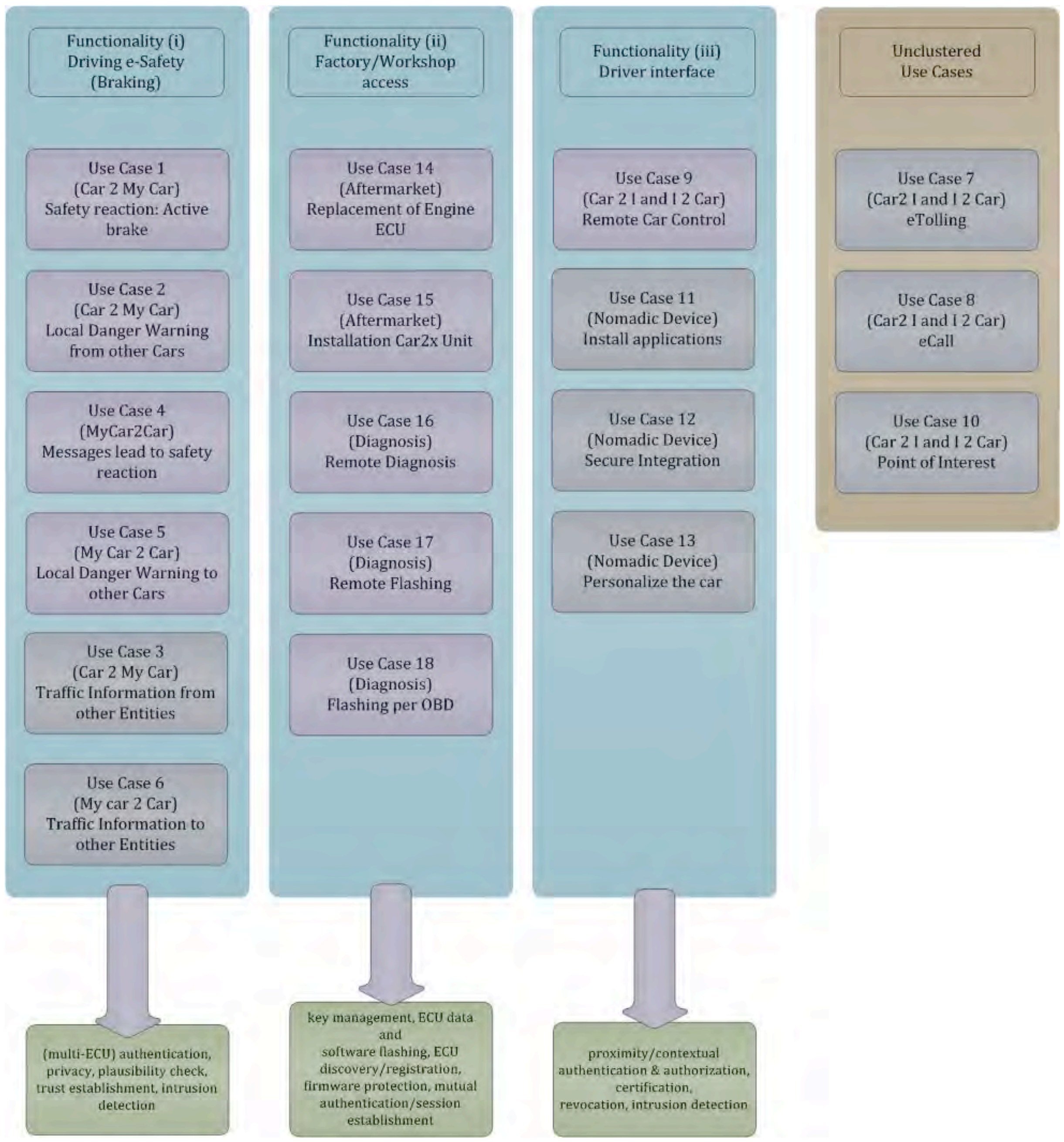

Functionality (i) Driving e-Safety (Braking)
Use Case 1 (Car 2 My Car) Safety reaction: Active brake
Use Case 2 (Car 2 My Car) Local Danger Warning from other Cars
Use Case 4 (MyCar2Car) Messages lead to safety reaction
Use Case 5 (My Car 2 Car) Local Danger Warning to other Cars
Use Case 3 (Car 2 My Car) Traffic Information from other Entities
Use Case 6 (My car 2 Car) Traffic Information to other Entities
(multi-ECU) authentication, privacy, plausibility check, trust establishment, intrusion detection
Functionality (ii) Factory/Workshop access
Use Case 14 (Aftermarket) Replacement of Engine ECU
Use Case 15 (Aftermarket) Installation Car2x Unit
Use Case 16 (Diagnosis) Remote Diagnosis
Use Case 17 (Diagnosis) Remote Flashing
Use Case 18 (Diagnosis) Flashing per OBD
key management, ECU data and software flashing, ECU discovery/registration, firmware protection, mutual authentication/session establishment
Functionality (iii) Driver interface
Use Case 9 (Car 2 I and I 2 Car) Remote Car Control
Use Case 11 (Nomadic Device) Install applications
Use Case 12 (Nomadic Device) Secure Integration
Use Case 13 (Nomadic Device) Personalize the car
proximity/contextual authentication & authorization, certification, revocation, intrusion detection
Unclustered Use Cases
Use Case 7 (Car2 I and I 2 Car) eTolling
Use Case 8 (Car2 I and I 2 Car) eCall
Use Case 10 (Car 2 I and I 2 Car) Point of Interest

Figure 1 Functional clusters of use cases

With the set of protocols we describe in this section, we cover different needs for security of applications and of underlying operating system and network requirements.

Considering the overall security requirements from [64], the most important security goals are:

- the integrity and authenticity of safety-relevant data,
- the integrity and authenticity of applications, e.g. running on ECUs, as well as their respective security configurations,

- the detection and containment of attacks,
- access control to data and functions as well as confidentiality of private information,
- confidentiality of certain data sent over the on-board network or sent to external entities,
- providing anonymity of personal information.

The specified protocols provide means to satisfy these security requirements. The protocols show how modules of the security framework described in [64] are used and how they interact. The HSM is abstracted primarily by the Cryptographic Services (CRS) module, which offers an API for various cryptographic functions. Some functions have not yet been available in CRS (e.g., for creating cryptographic keys). At these places, the HSM functions have directly been used accordingly[4].

The following protocols are addressed:

- Distribution of shared keys and certificates
  - A secure way to retrieve keys from external entities to the vehicle
  - A secure way to distribute keys between ECUs
- Session-keying and protocols for confidential and authentic communication
  - Establishment of session keys between ECUs
  - Session key establishment between key masters
- Platform Integrity and On-board integrity protocol
  - Remote attestation of ECUs and
  - Multipurpose ECUs
- Policy management and access control
  - Configuration of policies and policy-updates
  - Policy Synchronization
  - Access control protocols and policy enforcement
  - Firewall rules as part of access control policies
- Secure Bootstrapping
  - Synchronization of time
  - Synchronization of counters
  - Initialization of:
    - Session Keys and synchronization of Initialization Vectors (IVs).
    - Load secure configuration data, such as policies

[4]For the implementation, CRS is used after its specification has been revised.

- Further Security Management Protocols
  - Flashing: integrity check of firmware/code.
  - Maintenance: replacement of hardware components (includes key distribution/ key swapping).
  - Platform configuration update process for ECUs and sensors and corresponding firmware with KeyMaster. (including pairing/registration of nodes with KeyMaster)
  - Secure Device Integration
  - Secure Data Storage
- A Secure Transport Protocol

3.1.3 Use of Security Building Blocks

We consider the outcome of the security engineering process specified in [32] as input for the protocol specification. In [32] the security engineering process was applied to those use cases that match our newly defined use case cluster 1. The high level security requirements that were identified in [55] for these use cases were refined and the refined requirements were consolidated. Afterwards the consolidated requirements were categorized into three groups:

- **Internal** Requirements that are posed on a car by its driver,
- **C2C** Requirements that are concerned with the car-to-car communication,
- **Remote** Requirements that are posed on a car by other cars' driver or RSUs.

As stated in [32] requirements of the same category were expected to be addressable by similar, if not equal, Security Building Blocks (SeBBs) (Mechanism-SeBBs). Taking the consolidated *internal* requirements corresponding to use-case cluster 1 as example, we can derive that the further refinement could be achieved by e.g. using the Hash-Based MAC (HMAC), RSA-Sign or ECCSign Mechanism-SeBBs. In general we can derive that the main concern of messages of this type is authenticity and as a result a strong need for mechanisms, that ensure the authenticity of entities and data-origin, is present. In fact authenticity aspects are even more important than confidentiality aspects which leads to the strong focus on authenticity in the protocol specification in the following sections even to the extent that the EVITA transport protocol defined later aids the upper layer protocols by providing a way to integrate authenticity features even though the security features are not a direct component of the specified transport protocol.

3.1.4 Notation for Security Protocol Specification

We define abstract notations that refer to specific cryptographic methods. These notations are used throughout the document. In the following paragraphs, these notations are detailed.

We additionally list terms used to identify entities used in communication protocols.

Generation of MACs For generating a MAC as well as the message itself, the notation `Gen_MAC(k_mac,m,time_flag)` is used, so that it produces the message itself plus the cryptographic authentication code based on *k_mac* and m. Here, *k_mac* refers to a cryptographic key for MAC generation and m to the message to be authenticated. Based on the additional boolean flag `time_flag`, a time stamp is produced as an additional output parameter and then also covered by the MAC calculation. Regarding the cryptographic services specified in [64], this includes the sequence of the three function calls `CRS_generate_MAC_init`, `CRS_Update` and `CRS_Finish`. (Further notice: Although the time stamp is an additional optional value only present together with a MAC, it will always be listed before the MAC in the parameter sequence since it is also covered by the MAC generation. Furthermore, checking the freshness of the time stamp shall be possible before the MAC verification in case of networks with high latency.)

Data Encryption For generating an encrypted message, we use the notation `Encrypt(k_enc,m)`. The result will be a cryptogram `crypt` containing message m encrypted with the encryption key *k_enc*. Regarding the cryptographic services specified in [64], this includes the sequence of the three function calls `CRS_encrypt_init`, `CRS_Update` and `CRS_Finish`. This notation abstracts from concrete cryptographic algorithms used.

Data Decryption For decrypting an encrypted message, we use the notation `Decrypt(k_dec,crypt)`. As a result, a plaintext `plain` is produced from the cryptogram `crypt` with the decryption key *k_dec*. Regarding the cryptographic services specified in [64], this includes the sequence of the three function calls `CRS_decrypt_init`, `CRS_Update` and `CRS_Finish`. This notation abstracts from concrete cryptographic algorithms used. The cryptogram `crypt` is supposed to be the result of the encryption function described above.

EVITA application data The notation `EVITA_data` is used for data transport of applications. It is to be transferred without additional protocol header information but including the EVITA specific encoding of security features for the payload.

Data coding for message transfer The notation `Code_Data(data)` is used for the final coding of the string `data` as `EVITA_data`. For a correct coding of the security features, the calling application needs to know which of the previous functions `Gen_MAC` and `Encrypt` has been called before to create `data`. In case of MAC usage, the application also needs to know whether the whole MAC or only a fraction of it (see Section 3.1.5) shall be integrated.

Data Transfer The notation `Transfer_Data(payload_data)` shall be used for the transmission of a message containing the string `payload_data` (supposed to be of the type `EVITA_data` as described above) as payload data without further modification. This operation includes the embedding into transport protocol headers as well as fragmentation into several frames to be transmitted.

Data decoding after message transfer The notation `Decode_Data(payload_data)` is used for decoding the transferred message payload data (supposed to be of the type `EVITA_data`) into elementary data to which the functions `Decrypt` and `VER_MAC` may be applied. It also supplies information about the contained security features, i.e. which functions need to be applied by the calling application as well as the use of complete MACs vs. MAC fractions.

Detection and Separation of MACs For the determination of the original message m and its MAC (plus the additional time stamp if present) as separate parameters (especially for the purpose of verification), the notation `Det_MAC(data)` is used, whereby `data` is supposed to be concatenation of the message and its MAC produced by `Gen_MAC`.

Verification of MACs For the verification of a MAC, the notation `Ver_MAC(k_mac,m,time_stamp,MAC_data,time_flag)` is used. Based on the MAC key *k_mac*, it is verified whether `MAC_data` (being either a complete MAC or a fraction of it according to Section 3.1.5) corresponds to the message m and if the time stamp (denoted as third last parameter) is fresh. The message, its MAC as well as its time stamp (if applicable) are supposed to be delivered by the previously defined function `Det_MAC`. The boolean flag `time_flag` shall be set to the same value as in the previous corresponding call of `Gen_MAC`. If a time stamp was not used (`time_flag` = false), the time stamp parameter is ignored. Regarding the cryptographic services specified in [64], this includes the sequence of the three function calls `CRS_verify_MAC_init, CRS_Update` and `CRS_Finish`.

Key Master The notation `KeyMaster` is used for a special bus node that is used to establish communication. It is explained in detail in Section 3.2.1.

Signature This function is used for demonstrating the authenticity and integrity of a message. A valid signature gives a recipient reason to believe that the message was created by a known sender, and that it was not altered in transit. For signature generation, a signature generation scheme $sig(m)_{S_k}$ takes as input a key k, and message m, outputs a signature $\widehat{\sigma}$; we write $sig(\mathrm{m})_{S_k} = \{\widehat{\sigma}\}_{S_k}$. Where k is the security parameter, outputs a pair of keys $(S_k;V_k)$. S_k is the signing key, which is kept secret, and V_k is the verification key which is made public. We also assume that a time stamp (UTC Time) is generated, using `HSM_UTC_TIME(seconds_since_1970, mseconds_fraction)` function, and then also covered by the signature calculation, and write $\overrightarrow{m}$ = (`m + Ts`) to denote the message and a time stamp whose signature is $\widehat{\sigma}$. For the signature verification, *ver_-sig*$(\overrightarrow{m}\ , \widehat{\sigma})_{V_k}$ function is defined, takes as input the signature $\widehat{\sigma}$, the signature verification public key part V_k, and outputs the answer $\overrightarrow{m}$ which is either succeed (signature is valid) or fail (signature is invalid). As a precondition, the V_k must be loaded and enabled for verify.

3.1.5 Design Decisions: Fractions of Message Authentication Codes

Motivation and Approach One strategy for message size reduction is to use only a fraction of MACs instead of the whole MAC. Since we are dealing with symmetric keys, the verifier also needs the same key as the generator of the MAC. This means that we do not have to worry how to verify the fraction without knowing the whole and without knowing the key used to generate it (which would raise a problem in case of fractions of signatures based on public key cryptography). There are two fundamental approaches to achieve this:

- One approach is to first calculate the MAC and then to choose a part of the MAC, i.e., a partial bitstring.
- The other approach is to use intermediate or partial results of the MAC calculation process.

Since the calculations are performed in hardware, the time to execute cryptographic algorithms is no critical factor for us. We will use the first approach, i.e., truncate authentication codes, which better ensures the dependence of the MAC fraction on the whole key and does not require detailed cryptoanalyses. For verification, the receiver also first computes the whole MAC and then determines the corresponding fraction. This is no problem, since the receiver needs the complete symmetric MAC key anyway. Sender and receiver just need to agree in advance which part of the MAC shall finally be used as the fraction.

Conditions and Assumptions The security of the MAC fractions just depends on the length of the partial bitstring if the following conditions are satisfied:

- The whole MAC bitstring may be considered as random bitstring without different distributions for different bits, i.e. all bits randomly take the values 0 and 1 independent of each other.
- Each bit of the MAC string depends on each bit of the MAC key, i.e. any changes at the key may affect all bits of the MAC string.

These conditions may be satisfied if the MAC scheme provides the following properties:

- During MAC operation, the bits are mixed up in a way destroying any bit structure or arithmetic properties.
- There is no way to directly calculate the partial bitstring without the whole MAC key.

For HMAC schemes, the security properties directly depend on the underlying hash function since HMAC consists in application of hash function to data containing the key. For Cipher-Based MAC (CMAC) schemes, the security properties directly depend on the underlying symmetric encryption scheme since CMAC consists in the application of a cryptographic key.

Furthermore, we have chosen to concentrate on using 128 Bit Advanced Encryption Standard (AES) as symmetric encryption method and WHIRLPOOL (512 Bit) as cryptographic hash function. These functions may be considered as robust functions providing random strings where each output bit depends on each bit of the input and the cryptographic key. Therefore, we can assume that the above mentioned conditions are satisfied, i.e. we can evaluate the security of the MAC fractions just by their bitlengths.

After considering the final MAC bitstrings as random bitstrings, we now look at the success probabilities for random (brute force) attacks trying to find a preimage to a given signature or a collision (two preimages with the same MAC).

Concrete Impact on Security The complexity of random attacks finding a preimage to a given MAC (by just calculating MAC values to randomly chosen preimages) grows exponentially with the bitlength whereas the complexity of collision attacks grows exponentially with half the bitlength. This fact is well known for cryptographic hash functions (see e.g. [56]) but it also applies to other functions assumed to produce random strings equally dependent on all input bits. In other words, for an n bit MAC, you need to calculate about 2^n MACs to find a given preimage and about $2^{n/2}$ MACs to find a collision with a success probability of at least 50%.

In order to achieve a significant amount of data reduction, the MAC fraction should not be longer than half the original MAC. Table 2 summarizes the quantitative results for using half or a quarter of an AES CMAC, an SHA-256 HMAC or a WHIRLPOOL HMAC.

MAC Type	Bit length	Number of guesses for	
		random attack	square-root attack
Full WHIRLPOOL HMAC	512	2^{512}	2^{256}
Half WHIRLPOOL HMAC	256	2^{256}	2^{128}
Quarter of WHIRLPOOL HMAC	128	2^{128}	2^{64}
Full SHA-256 HMAC	256	2^{256}	2^{128}
Half SHA-256 HMAC	128	2^{128}	2^{64}
Quarter of SHA-256	64	2^{64}	2^{32}
Full AES CMAC	128	2^{128}	2^{64}
Half AES CMAC	64	2^{64}	2^{32}
Quarter of AES CMAC	32	2^{32}	2^{16}

Table 2 Expected effort for brute force and collision attacks for MAC fractions

Facit: For a 512 bit HMAC based on WHIRLPOOL, using just half or even only a quarter of the complete signature still provides sufficient security. In case of a 256 bit HMAC (e.g. based on SHA-256), we can go down to half the MAC length for still withstanding a square root attack and to a quarter of the MAC length for withstanding a random attack. Using half the bitlength for an AES-CMAC provides enough security for withstanding random attacks, but not for withstanding square root (collision) attacks since 2^{32} is not an infeasible number of trials for an attacker. According to Annex A of [15], fractions of 64 bits, i.e. half an AES-CMAC guarantee sufficient protection against guessing attacks without further restrictions. The FIPS recommendation on this topic [61] points out that the length can only be decreased further, if additional measures are taken. In our application domain, we have rather slow buses and can additionally limit the rate of false verifications of authentication codes, so that further decreasing the length to 32 bits is feasible.

In our application environment, square root attacks (with possible modifications of genuine messages that will always be signed) are impractical since any message to be secured with a MAC is internally created by the system. Therefore, we only have to concentrate on random attacks. This means that shortening the final MAC results to 64 bits is an acceptable solution, i.e., we can still use half of the MAC if we only have available the functionality to work with AES-CMAC. As recommended by NIST [15], shorter MAC lengths could be allowed if additional security features are taken into account. An additional feature would be limiting the rate of negative MAC verifications on the HSM (for example to a maximum of ten verification failures per second for one key). As this creates a possible denial-of-service weakness, it has to be considered carefully. By combining this measure with others of the security framework (in particular Security Watchdog Module (SWD) plug-ins), a compromise may be found. Possible countermeasures such as a simple challenge-response nonce could be added to the payload to avoid denial-of-service attacks. Such countermeasures would of course only be taken if a possible attack (i.e., a high rate of invalid MACs) is detected.

3.2 Key Distribution Protocols

3.2.1 Key Master Architecture

For presenting our protocols on selected use cases, we use the EVITA reference architecture, equipped with different sorts of HSMs such as suggested in the deployment chapter of [64].

We introduce a new functional entity, which we call the "KeyMaster". This functional entity may reside on a dedicated ECU or be integrated into another ECU. It is used to hold Pre-Shared Keys (PSKs), shared with individual ECUs, see Figure 2. This is used to enable secure group communication in a Kerberos-like fashion, i.e. an authentication server holds shared secret key material.

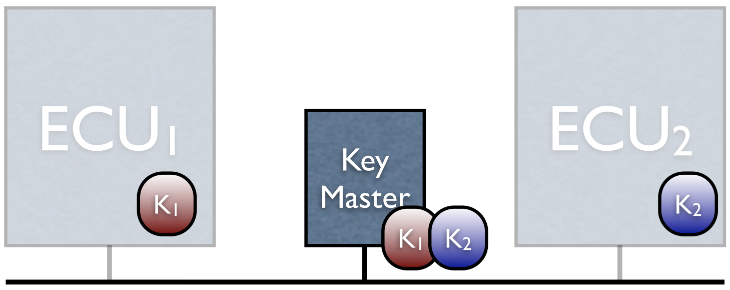


Figure 2 KeyMaster entity as separate node

There may be more than one KeyMaster node in a vehicle. KeyMasters may be replicated in different locations. They may also be deployed in a per-domain manner, so that key information is not shared outside a specific domain.

Distributing cryptographic key material is a key issue for secured communication. In vehicles, we have different types of communication, of which most is signal-based and a sender node serves multiple receiving nodes. We call these a *group*. A group usually consists of one sending node and multiple receivers. Following our HSM approach, where multiple key-properties may be set, only sending nodes may sign and encrypt data, whereas receiving nodes can verify and decrypt data using the same key material.

As there exist multiple variants of the HSM, including the EVITA light HSM that only supports symmetric cryptography, we had to take this into account for key distribution. It has an impact on the architecture, so that a trusted third party is needed within the vehicle because key material has to be shared between at least two entities.

We take the following scenarios into account:

- Only full and medium HSMs in the communication group
- At least one light HSM in one group
- At least one ECU without HSM in the group.

Pre-shared keys with a so called "key-master node" (referred to as KeyMaster or KM) are needed for cases in which light HSMs are used, as asymmetric encryption could not be performed in hardware, i.e., key material could not be protected appropriately. Keys are transferred between ECUs using the HSM's migration feature, so that temporary (session) keys are encrypted using the pre-shared secret.

If keys are pre-distributed and never changed during the lifetime of the vehicle, such a key-master node may not be needed. In compliance with current standards, we anticipate a limited lifetime of key material. Such a session is envisaged to last for one drive cycle or a maximum of 48 hours, after which it is not valid for use anymore.

Figure 3 shows session keys being shared by a number of nodes, as well as the KM. This session key refers to the cryptographic key, as the key-structure itself differs in its "use flags": Only one node (ECU_1) may sign data, whereas the others may only verify data using the key. This is adaptable to confidential group communication (i.e., use-flags encrypt/decrypt) analogously. The sign (or

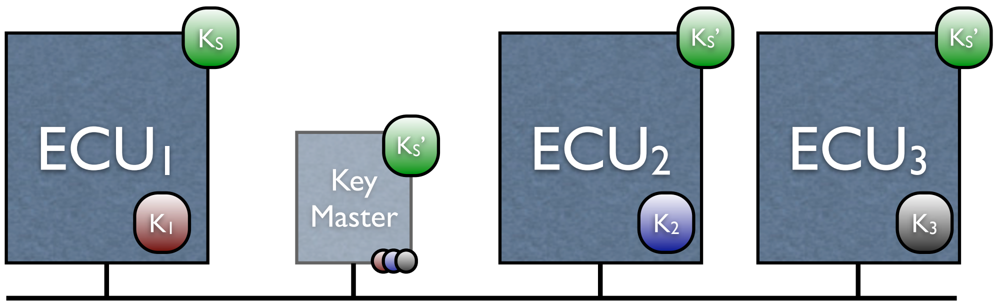


Figure 3 Session keys shared by a group of ECUs. Generation Key K_s stays at ECU_1 and verification key K_s' is exported to other ECUs via the KeyMaster node. A PSK (depicted at the bottom of the ECUs) is shared between an ECU and the KeyMaster.

3.2.2 Keying using asymmetric encryption

Figure 4 shows the sequence diagram of a session key establishment between two ECUs. This type of communication is only feasible if both ECUs are equipped with at least the medium HSM (i.e., asymmetric cryptography can be performed inside the HSM and doing it in software is not feasible with regard to security). The session establishment shown does not take a trusted signature into account and assumes the KeyMaster as a trusted third party that stores and validates public keys of internal ECUs. If the ECUs' Device Identity Keys (IDKs) are signed with the same Manufacturer Verification Key (MVK), this step can be omitted. The KeyMaster is used as reference of authenticity of the public keys. If certificates issued by a trusted third party are used (e.g., the Public-Key Infrastructure (PKI) of the manufacturer whose key is referred to as MVK), this extra communication can be omitted, as the trust anchor is then located outside the vehicle. For this scenario, the interaction with the HSM has been described in detail in [64]. In any case, the IDK shall only be used in authenticated encryption mode. If the specific algorithm does not allow for this mode, two keys per ECU need to be exchanged, of which one is used for signing and the other for encrypting data.

3.2.3 Keying for group communication: symmetric approach

Figure 5 shows how one sender ECU_1 may establish a secure group with receiving nodes $\text{ECU}_2 \ldots \text{ECU}_n$. This is a one-to-many communication use case as it is used to physically broadcast signals in today's vehicles (at the logical level, it is a multicast connection). ECU_1 is the sending partner of the communication group. The KeyMaster is a central element in the establishment of the session. It holds pre-shared keys of the individual ECUs, which are used as transport keys (TK) in the establishment process. ECU_1 creates a session-key pair[5] with a key used to generate MACs and to verify MACs (exportable). The sign key (or encrypt, respectively) can not be exported, as it is created without the transport flag. The exportable key is sent in an Open Channel request to the KeyMaster node. It is exported enciphered with the pre-shared transport key.

The KeyMaster verifies authenticity and authorization of ECU_1 using the Policy Decision Module (PDM) (i.e., whether ECU_1 is allowed to establish group communication to group g_x as sender). It then distributes the session key to the individual ECU participating in group g_x. The session key is encrypted with the corresponding PSK, as it is used as transport key for the HSM. Once these keys have been imported, acknowledgments are sent back to the KeyMaster. After all acknowledgments have been received by the KeyMaster, it tells ECU_1 that the group is now equipped with the session key (last ACK in diagram). After this acknowledgment frame, ECU_1 can send secured (authentic) data to participants of the group. For confidentiality, key properties would need to be changed to (encrypt,decrypt) for (k,k'). For details, please see the Communication Control Module (CCM) section in [64] and its corresponding references.

Please note: Data structures used in for implementation need to take the role of individual ECUs in different groups into account, i.e., whether an ECU initiates or only passively participates in a communication group.

[5]We use the term *Key Pair* for those symmetric keys that feature complementary functions (sign/verify) and (encrypt/decrypt).

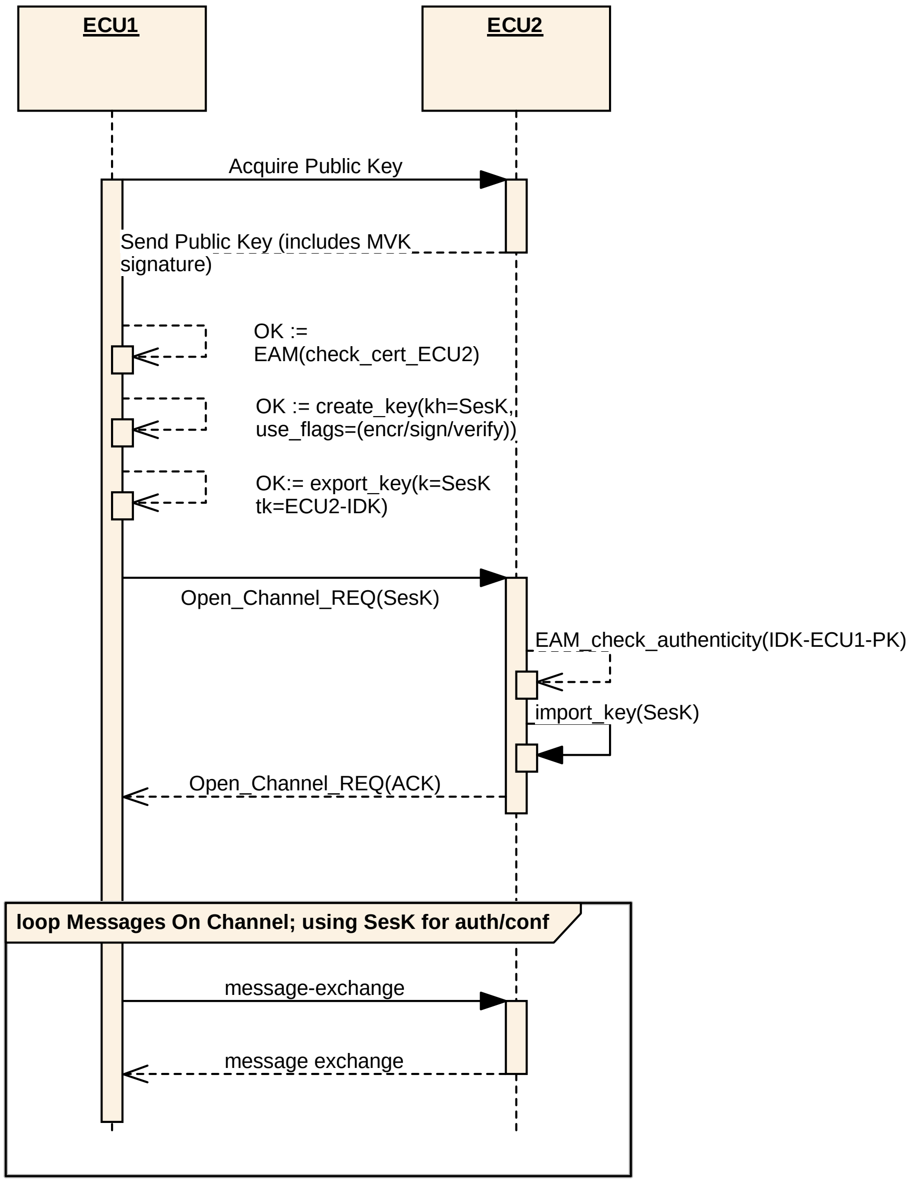

ECU1
ECU2
Acquire Public Key
Send Public Key (includes MVK signature)
OK := EAM(check_cert_ECU2)
OK := create_key(kh=SesK, use_flags=(encr/sign/verify))
OK:= export_key(k=SesK tk=ECU2-IDK)
Open_Channel_REQ(SesK)
EAM_check_authenticity(IDK-ECU1-PK)
import_key(SesK)
Open_Channel_REQ(ACK)
loop Messages On Channel; using SesK for auth/conf
message-exchange
message exchange

Figure 4 Session key establishment between two ECUs. No KeyMaster needs to be involved, as both device public keys (IDK-PK) are signed by the manufacturer (MVK), i.e., the ECUs carry certificates. The IDK-PK shall be used in authenticated encryption mode. Interaction of internal modules is omitted; further details can be found from following sequence diagrams and in the annex.

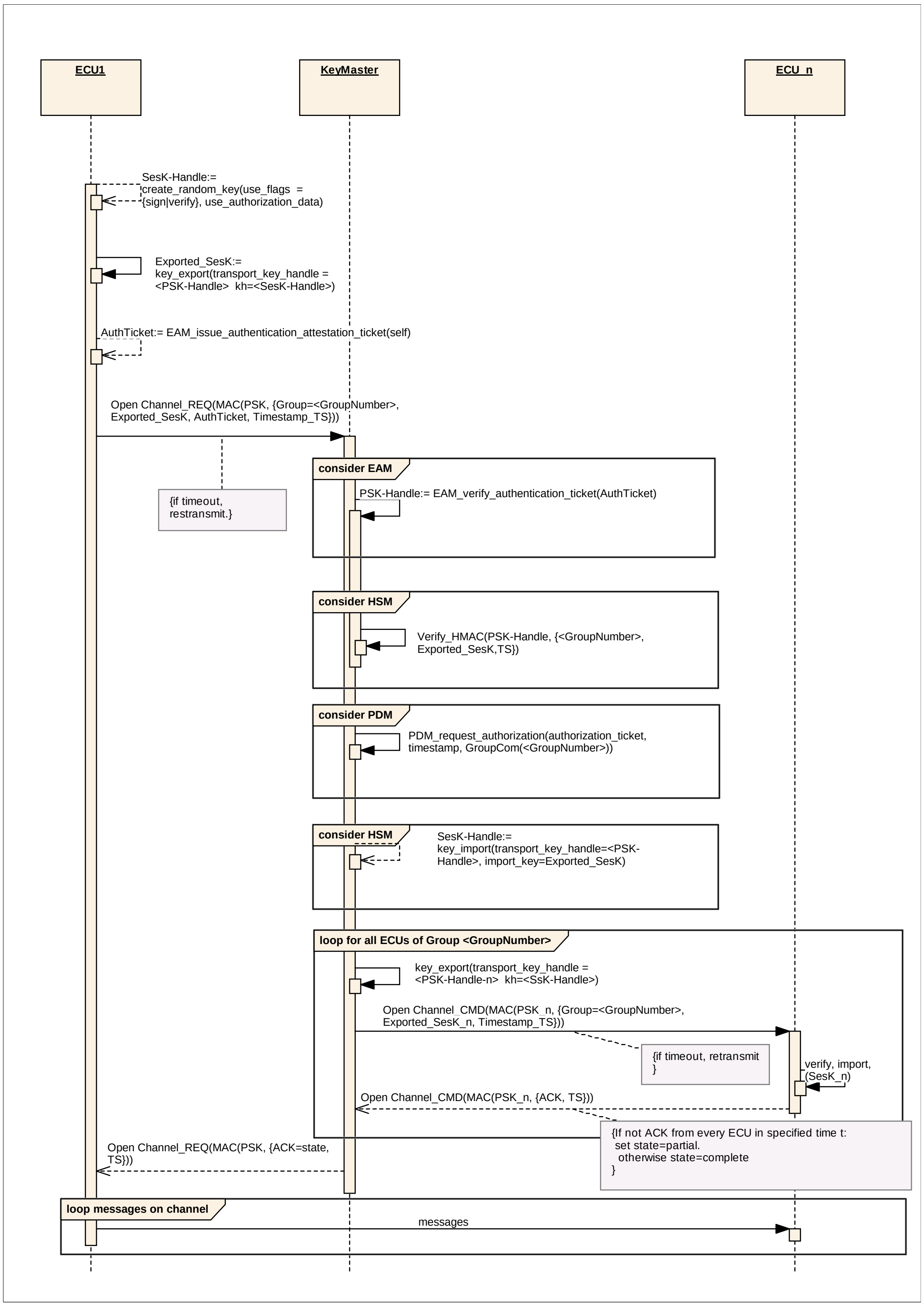

ECU1
KeyMaster
ECU_n
SesK-Handle:= create_random_key(use_flags = {sign|verify}, use_authorization_data)
Exported_SesK:= key_export(transport_key_handle = <PSK-Handle> kh=<SesK-Handle>)
AuthTicket:= EAM_issue_authentication_attestation_ticket(self)
Open Channel_REQ(MAC(PSK, {Group=<GroupNumber>, Exported_SesK, AuthTicket, Timestamp_TS}))
{if timeout, restransmit.}
consider EAM
PSK-Handle:= EAM_verify_authentication_ticket(AuthTicket)
consider HSM
Verify_HMAC(PSK-Handle, {<GroupNumber>, Exported_SesK,TS})
consider PDM
PDM_request_authorization(authorization_ticket, timestamp, GroupCom(<GroupNumber>))
consider HSM
SesK-Handle:= key_import(transport_key_handle=<PSK-Handle>, import_key=Exported_SesK)
loop for all ECUs of Group <GroupNumber>
key_export(transport_key_handle = <PSK-Handle-n> kh=<SsK-Handle>)
Open Channel_CMD(MAC(PSK_n, {Group=<GroupNumber>, Exported_SesK_n, Timestamp_TS}))
{if timeout, retransmit }
verify, import, (SesK_n)
Open Channel_CMD(MAC(PSK_n, {ACK, TS}))
{If not ACK from every ECU in specified time t: set state=partial. otherwise state=complete }
Open Channel_REQ(MAC(PSK, {ACK=state, TS}))
loop messages on channel
messages

Figure 5 Session key establishment for group communication with symmetric PSK. A key pair is generated. The sign (or encrypt) key can not be exported and rests in the HSM. The verify (or decrypt) key is exported and transmitted via the KM to other communication partners. The initiated communication takes place in a *one-to-many* fashion.

3.3 Platform integrity checks for multi-purpose ECUs

3.3.1 Overview

Key security requirements of safety applications and safety related information are integrity, authenticity, and trustworthiness. Applying virtualization techniques on multipurpose ECUs supports these security requirements, as this technology provides the separation of processes and applications, and hence allows for the execution of applications with different trust levels on the same ECU. As application domains with varying trust levels may communicate within the on-board network, it is reasonable to additionally provide respective security measures in order to control the information flow amongst such domains. We believe that the combination of virtualization and Trusted Computing technologies (e.g., based on TPMs, or EVITA HSMs) provides the measures for meeting our discussed security requirements.

3.3.2 Scenarios

We consider two different scenarios that are based on trends of on-board vehicular networks as described in [64]. Within the first scenario, we presume the co-existence of multipurpose ECUs and common ECUs, whilst within the second scenario we consider an on-board architecture where only multifunctional ECUs are deployed. The first scenario can be seen as a first step of evolution where not all ECUs are ported to multipurpose ECUs, but also self-contained, widely autonomous ECUs still exist in the on-board infrastructure. The next step in evolution could be that all ECUs are ported to multipurpose ECUs. Figure 6 shows a multipurpose ECU communicating with other common ECUs. The multipurpose ECU consists of a virtualization-supporting hardware platform, such as Intel's Atom processor, a hypervisor for virtualizing the underlying hardware, and a number of Virtual Machines (VMs). In addition, we assume an automotive-capable hardware-based trust anchor, such as the EVITA HSMs. Each VM is strongly isolated from other VMs and executes the software environment of a proprietary ECU. In addition, the hypervisor provides mechanisms such as virtual machine inspection that allows monitoring the ECU VMs. Hence, if it detects a state that is considered insecure or untrusted, it can reset the VM to a known secure state.

3.3.3 Integrity Stage Checks

Integrity stage checks ensure that when a specific stage has been successfully passed, the platform satisfies a set of specific security requirements associated to a specific stage, thus satisfying the requirement that the integrity of on-board ECU platforms can be reliably enforced or modified platform configurations can at least be reliably detected. Integrity stage checks complement the secure boot process by extending the secure boot process over the virtualization boundary and by applying more fine-granular checks. The integrity stage checks performed in our architecture are as follows:

Integrity stage check 1: security controller The first integrity stage check is performed after successful execution of the security controller. For this purpose, the security

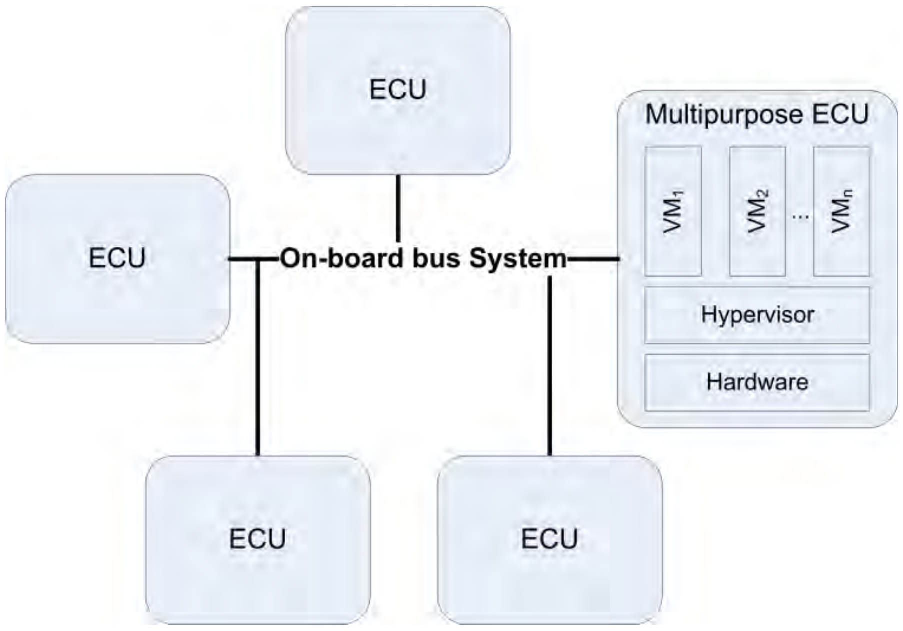


Figure 6 Multipurpose ECU communicating with other ECUs

controller's virtual appliance is sealed to the Platform Configuration Register (PCR) values using a key that resides in the HSM, for instance the Storage Root Key (SRK). The unsealing of the security controller is initiated by the secure boot which measures the VM bootloader and hands over control to the bootloader. The bootloader then unseals the virtual appliance of the security controller and spawns the security controller. As a result, the first integrity stage check can only be passed if the PCR values are in a known state and the virtual appliance of the security controller can be decrypted.

Integrity stage check 2: vTPM-storage The second integrity stage check can only be passed if the vTPM storage of a specific vTPM instance is in a known and authentic state. For this purpose, the vTPM storage is sealed to the PCR of the HSM using the SRK. Note that this approach requires updating and a resealing of the vTPM persistent storage each time new data, such as cryptographic keying material, is placed inside this secure storage of the vTPM.

Integrity stage check 3: ECU specific attestation key If the preceding integrity stage checks have been successfully passed, the ECU has access to its own associated vTPM. The last integrity stage check is performed before an ECU is able to use his ECU specific key K_{ECU}. This key is bound to the virtual TPM's PCR and thus only usable if all previous checks succeed and the vTPM's PCRs are the same as when K_{ECU} was initially bound to. In addition, access to this K_{ECU} is only possible if the security controller is running and, thus, is able to validate all in-vehicle messages originating in an ECU.

3.3.4 Attestation Protocols

To enable attestation, we divide into an initialization phase and an attestation phase. In the initialization phase, which is typically executed only once, the vTPM is equipped with a special key (K_{ECU}) which is later used for attestation and which is bound to the configuration of the VM. This key is then certified by a trusted party to ensure that the ECU's configuration is known and trusted. The advantage of this concept is that an ECU is able to prove to another entity that it is trusted without requiring the other entity to perform complex computations to evaluate the trustworthiness of the ECU.

Initialization Phase with Certification Protocol When the software configuration of the ECU changes, a separate certification protocol needs to be executed. Its purpose is to generate a cryptographic key which directly identifies an ECU and is only usable if the software configuration is in the same state as this cryptographic key was initially bound to. The protocol for issuance of such a key and the corresponding certificate is described in the following, where V denotes a validator that is able to validate the platform integrity, e.g., the supplier of the ECU firmware, ECU is an ECU that wants to receive a certificate, and SC is the security controller. First, V must acquire (and validate) the certificate of the Privacy-CA to validate Cert(vAIK, K_{vAIK}). The protocol can be executed over an insecure channel which allows for remotely updating and integrating new ECU components on a multipurpose hardware platform. The protocol is shown in Figure 7.

Attestation Phase After the successful initialization phase, the ECU is now in possession of K_{ECU} and the corresponding certificate. The key K_{ECU} can now be used to prove to another ECU that it is trusted, by simply signing a fresh message with this key. The advantage of this concept is that a verifier does only need to verify whether the certificate is valid, rather than parsing the whole measurement chain. The resulting protocol is shown in Figure 8.

3.4 Policy Management and Access Control Protocols

Security policies constitute an important part in the EVITA security software framework in that they constitute a description of an interface with the vehicular network behavior for security administrators at the different stakeholders. Security policies will be uploaded to the car, and in particular to PDMs, before the vehicle can run. These policies can be uploaded off-line or updated remotely.

3.4.1 Security Policies

Security policies consist mainly of access control policies and intrusion detection policies.

- Access control policies are aimed at controlling which stakeholder can access which functionality of an ECU. These policies are subdivided into filtering policies and endpoint access control policies, depending on the location of enforcement of the access control decision.

1. *Protocol messages of the certification phase.*

$$ECU \rightarrow SC : \mathit{Cert(vAIK,\ K_{vAIK})},\ \mathit{SML},\ \{\mathit{Cert(ECU,\ K_{ECU})},\ \mathit{Assertion}_{TPM\_PCR\_INFO}\}_{K^{-1}_{vAIK}} \quad \text{(I)}$$

$$SC \rightarrow V : \mathit{Cert(vAIK,\ K_{vAIK})},\ \mathit{SML},\ \{\mathit{Cert(ECU,\ K_{ECU})},\ \mathit{Assertion}_{TPM\_PCR\_INFO}\}_{K^{-1}_{vAIK}} \quad \text{(II)}$$

$$V \rightarrow SC : \{\{timestamp, Certifier, ECU_{prop}, K_{ECU}\}_{K_V^{-1}}\}_{K_{ECU}} \quad \text{(III)}$$

$$SC \rightarrow ECU : \{\{timestamp, Certifier, ECU_{prop}, K_{ECU}\}_{K_V^{-1}}\}_{K_{ECU}} \quad \text{(IV)}$$

2. *Protocol actions.*

   (a) *Precomputation and Pre-deployment by ECU. ECU* creates a non-migratable vTPM key (K_{ECU}) that is bound to a specific set of PCRs. *ECU* certifies K_{ECU} with K^{-1}_{vAIK}. The resulting structure is denoted $\{\mathit{Cert(ECU,\ K_{ECU})},\ \mathit{Assertion}_{TPM\_PCR\_INFO}\}_{K^{-1}_{vAIK}}$ and includes a `TPM_PCR_INFO` structure which gives an assertion to which PCR K_{ECU} is bound to. In effect, this structure contains all vTPM PCR values. The resulting certificate is then delivered with the stored measurement log (SML) and the certificate of the *vAIK* in message (I) to *SC*.

   (b) *SC validation.* The security controller validates based on the policy if the ECU is allowed to perform an initialization phase. If this step succeeds, *SC* forwards message (II) to *V*.

   (c) *Integrity validation.* *V* checks whether the ECU is trusted. For this purpose, *V* processes the *SML* and re-computes the received PCR values. If the computed values match the signed PCR values, the *SML* is valid and untampered. In addition, the verifiers checks if the used *Cert(vAIK, K_{vAIK})* was derived from a genuine and valid *AIK*. If all steps succeed, *V* creates a certificate $\{\mathit{timestamp,\ Certifier,\ ECU_{prop}}, K_{ECU}\}_{K_V^{-1}}$ which states that K_{ECU} is bound to a trusted platform configuration. ECU_{prop} is a data structure where all platform characteristics of the ECU, i.e., properties, are encapsulated. This data structure includes information about the type of ECU, its trust level, and other characteristics. This certificate is sent in message (III) to the security controller that forwards the message to the ECU (IV).

Figure 7 ECU Certification Protocol for Initialization Phase

1. *Protocol messages of the attestation phase.*

$$ECU_{source} \rightarrow SC_{source} : \{timestamp,\ Certifier,\ ECU_{prop}, K_{ECU}\}_{K_V^{-1}},\ message,\ \{message\}_{K_{ECU}^{-1}} \quad (1)$$

$$SC_{source} \rightarrow SC_{dest} : \{timestamp,\ Certifier,\ ECU_{prop}, K_{ECU}\}_{K_V^{-1}},\ message, \{message\}_{K_{SC_{source}}^{-1}},\ \{message\}_{K_{ECU}^{-1}} \quad (2)$$

$$SC_{dest} \rightarrow ECU_{dest} : message, \{message\}_{K_{SC_{dest}}^{-1}} \quad (3)$$

2. *Protocol actions.*

   (a) *Message creation.* ECU_{source} loads the K_{ECU}^{-1} from the vTPM storage into the vTPM and signs the message that is to be transmitted to ECU_{dest} with K_{ECU}^{-1}. The message includes all information that is necessary to deliver this message to the correct ECU_{dest}, i.e., *message* includes information about the source ECU and the destination ECU. In addition, the message includes a random number to prevent replay attacks. It is thus of the following form:

$$message = ID\_ECU_{source}, ID\_ECU_{dest}, rand, msg \quad (1)$$

   The message is then delivered in message (1) together with $\{timestamp,\ Certifier,\ ECU_{prop}, K_{ECU}\}_{K_V^{-1}}$ to the source security controller.

   (b) SC_{source} validates based on the policy whether ECU_{source} is allowed to deliver messages to ECU_{dest}. If this is true, SC_{source} signs the message with $K_{SC_{source}}^{-1}$ and transmits message (2) to SC_{dest}. In addition, this message can also be encrypted with the destination's ECU bound public key to ensure that only an ECU which is in a trusted state can decrypt this message.

   (c) *Integrity validation.* SC_{dest} validates whether the certificate $\{timestamp,\ Certifier, ECU_{prop}, K_{ECU}\}_{K_V^{-1}}$ is valid. In addition, SC_{dest} verifies based on the certified platform characteristics (ECU_{prop}) whether ECU_{source} is trusted and whether ECU_{source} is allowed to deliver messages to SC_{dest}. If the checks succeed, SC_{dest} signs the message with $K_{SC_{dest}}^{-1}$ and delivers message (3) to the destination ECU.

   (d) ECU_{dest} verifies whether $\{message\}_{K_{SC_{dest}}^{-1}}$ is authentic and fresh by validating whether the integrated random number was not already used in the past. Finally, ECU_{dest} processes the message.

Figure 8 ECU Attestation Protocol

  - Filtering policies are enforced at gateways, which act as firewalls. The aim of those policies is to decide whether to forward messages or to drop them based on the authorization rules defined. Those rules will typically define patterns that have to be matched by the transport layer for the message to be forwarded (positive rules) or dropped (negative rules). Every gateway will contain a list of such rules that will be screened in an orderly fashion (so as to solve conflicts between positive and negative rules), looking for the first match. Rules apply to the incoming interface. The case where no match applies will result in the message being dropped, and a notification being sent to the intrusion detection system. Rules of such policies will be based mainly on transport-layer parameters (source and destination addresses, domain of origin or destination, etc.) but also on application-layer information, resulting for instance not only from the observation of a message (e.g., sending of a message from a sensor), but also on that part of its content (depending on the processing capabilities of the gateway), e.g. contained security features on application level (see Section 3.8). Filtering may also be stateful, for instance relying on the past observation of a message, and even on contextual information, i.e., information about the vehicle and its environment that parameterize the policy but may be updated separately (e.g., vehicle speed, nearby vehicles, attack detected on another gateway, etc.). Filtering may therefore rely on plausibility checks. Contrary to filtering performed for traffic coming from some domains (e.g., the Head Unit (HU) domain), filtering on emergency messages should never completely prevent the transmission of potentially e-safety related messages, and only endpoints, i.e., applications, should ultimately decide about the validity of some information.
  - Endpoint access control policies on the other hand determine precisely whether an application can process certain messages based on its origin and on the definition of the authorizations of stakeholders. These are again positive rules expressing authorization granted to subjects. These policies are rather fine-grained. Most of these policies are about application permissions, that is, operations that can be performed by ECUs internally. However, some permissions relate to stakeholder's rights, in particular with respect to the right to update firmware or parameters of ECUs. Finally a meta-access control policy describes which stakeholders are allowed to update other access control policies and filtering policies.

- Intrusion detection policies define the behavior of the security watchdog. They aim at expressing abnormal situations that should not arise in the vehicle, combined with another series of plausibility checks. These policies rely on a set of rules consisting of a pattern and an action. Matching the pattern should trigger the action. Actions may consist in updating the contextual information available at PDMs, i.e. updating the state of the PDM, or even updating the policy of PDMs (filtering or endpoint access control), i.e., reconfiguring its access control rules. Both needs are addressed by the policy configuration protocol (see below). Finally, as mentioned above, actions that might result in dropping e-safety related messages might potentially be aborted by PDMs.

3.4.2 Policy Update Protocol

The design of secure policy update for on-board architecture can become quite complex and error-prone, especially when information flows from different trusted to untrusted domains and vice versa. Updating the security policies (basically access control) on the air in vehicles requires a secure procedure which can be achieved by using various cryptographic techniques. However, due to the nature of the on-board architecture, more specialized and customized protocols are designed for the automobile scenario. In order to provide secure policy update protocol, we listed the most desired security properties *Authentication, Integrity, Freshness, Non-Repudiation, Anonymity, Availability, and Confidentiality* to be able to prevent threats, as identified in [33]. These properties ensure that policies are deployed/stored securely in the vehicle. The suggested protocol is illustrated in Figure 9.

Protocol Description: Security policy updates are routinely provided on the "`X`" time duration (i.e. provided after a specified time has passed), but can be provided whenever a new update is urgently required to prevent a newly discovered or prevalent exploit targeting on-board systems. It is assumed that only updates that have the approval status *Install* will be downloaded to the vehicle. By default, critical and security updates are already approved for detection (Detect Only), which means the policy server will determine if these updates are needed by the vehicle. These updates will still need to be approved for install before downloading. All other new updates will show up as *Not Approved* until the vehicle owner decides to approve them for install or decline them.

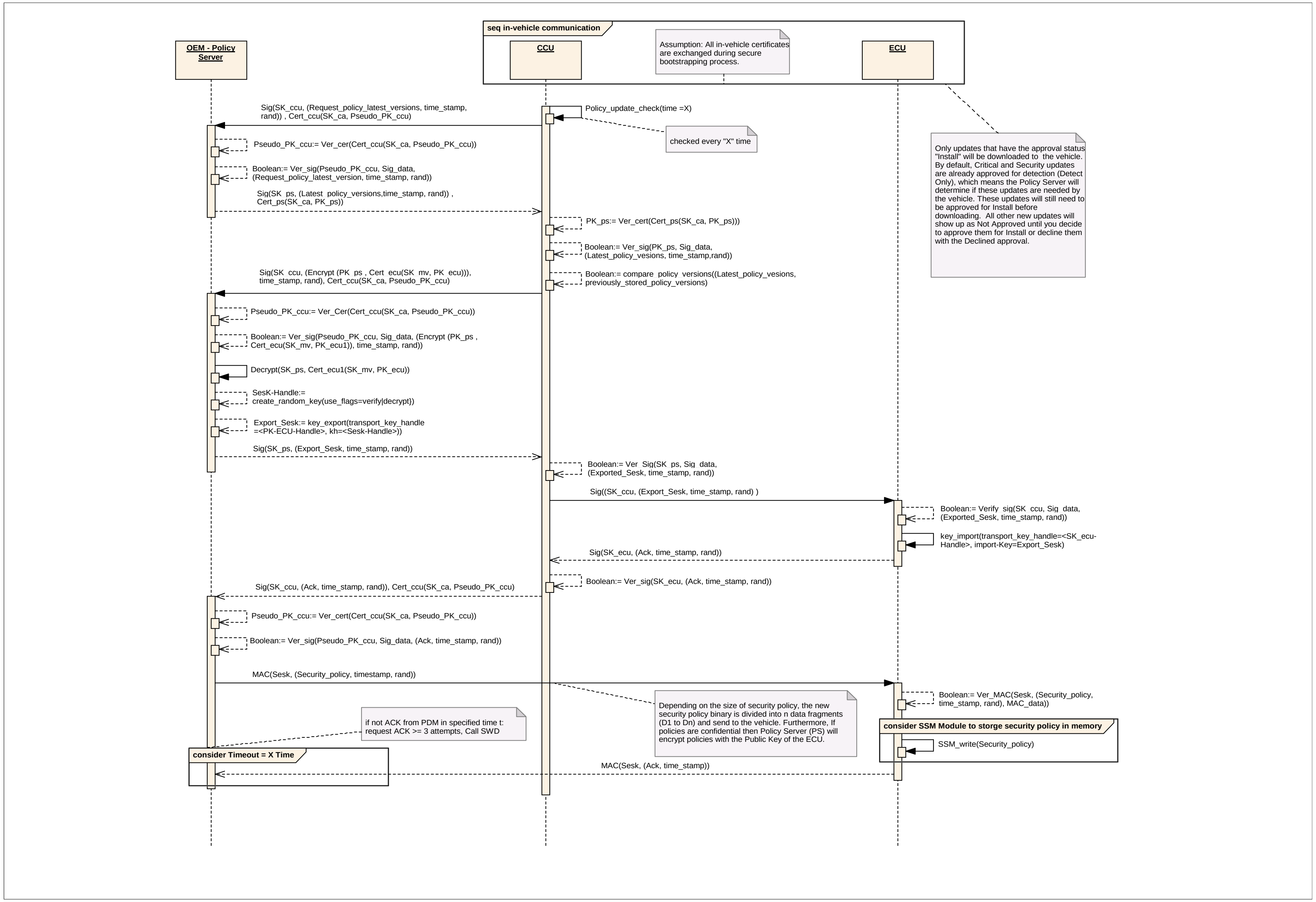

seq in-vehicle communication
OEM - Policy Server
CCU
ECU
Assumption: All in-vehicle certificates are exchanged during secure bootstrapping process.
Policy_update_check(time =X)
checked every "X" time
Only updates that have the approval status "Install" will be downloaded to the vehicle. By default, Critical and Security updates are already approved for detection (Detect Only), which means the Policy Server will determine if these updates are needed by the vehicle. These updates will still need to be approved for Install before downloading. All other new updates will show up as Not Approved until you decide to approve them for Install or decline them with the Declined approval.
Sig(SK_ccu, (Request_policy_latest_versions, time_stamp, rand)) , Cert_ccu(SK_ca, Pseudo_PK_ccu)
Pseudo_PK_ccu:= Ver_cer(Cert_ccu(SK_ca, Pseudo_PK_ccu))
Boolean:= Ver_sig(Pseudo_PK_ccu, Sig_data, (Request_policy_latest_version, time_stamp, rand))
Sig(SK_ps, (Latest_policy_versions,time_stamp, rand)) , Cert_ps(SK_ca, PK_ps))
PK_ps:= Ver_cert(Cert_ps(SK_ca, PK_ps)))
Boolean:= Ver_sig(PK_ps, Sig_data, (Latest_policy_vesions, time_stamp,rand))
Boolean:= compare_policy_versions((Latest_policy_vesions, previously_stored_policy_versions)
Sig(SK_ccu, (Encrypt (PK_ps , Cert_ecu(SK_mv, PK_ecu))), time_stamp, rand), Cert_ccu(SK_ca, Pseudo_PK_ccu)
Pseudo_PK_ccu:= Ver_Cer(Cert_ccu(SK_ca, Pseudo_PK_ccu))
Boolean:= Ver_sig(Pseudo_PK_ccu, Sig_data, (Encrypt (PK_ps , Cert_ecu(SK_mv, PK_ecu1)), time_stamp, rand))
Decrypt(SK_ps, Cert_ecu1(SK_mv, PK_ecu))
SesK-Handle:= create_random_key(use_flags=verify|decrypt})
Export_Sesk:= key_export(transport_key_handle =<PK-ECU-Handle>, kh=<Sesk-Handle>))
Sig(SK_ps, (Export_Sesk, time_stamp, rand))
Boolean:= Ver_Sig(SK_ps, Sig_data, (Exported_Sesk, time_stamp, rand))
Sig((SK_ccu, (Export_Sesk, time_stamp, rand))
Boolean:= Verify_sig(SK_ccu, Sig_data, (Exported_Sesk, time_stamp, rand))
key_import(transport_key_handle=<SK_ecu-Handle>, import-Key=Export_Sesk)
Sig(SK_ecu, (Ack, time_stamp, rand))
Boolean:= Ver_sig(SK_ecu, (Ack, time_stamp, rand))
Sig(SK_ccu, (Ack, time_stamp, rand)), Cert_ccu(SK_ca, Pseudo_PK_ccu)
Pseudo_PK_ccu:= Ver_cert(Cert_ccu(SK_ca, Pseudo_PK_ccu))
Boolean:= Ver_sig(Pseudo_PK_ccu, Sig_data, (Ack, time_stamp, rand))
MAC(Sesk, (Security_policy, timestamp, rand))
Boolean:= Ver_MAC(Sesk, (Security_policy, time_stamp, rand), MAC_data))
Depending on the size of security policy, the new security policy binary is divided into n data fragments (D1 to Dn) and send to the vehicle. Furthermore, If policies are confidential then Policy Server (PS) will encrypt policies with the Public Key of the ECU.
consider SSM Module to storge security policy in memory
SSM_write(Security_policy)
if not ACK from PDM in specified time t: request ACK >= 3 attempts, Call SWD
consider Timeout = X Time
MAC(Sesk, (Ack, time_stamp))

Figure 9 Remote Policy Update Protocol

1. *Policy Update Check.* A `Policy_update_check` function is defined and deployed on the Communication Control Unit (CCU) that checked the policy update on a regular schedule for new updates that have been marked as *Critical* or *Complimentary.* By default, this check occurred every "`X`" time. However, it is still reasonable to consider that `Policy_update_check` function is deployed on other ECUs too. The check is performed by querying the policy server for policy updates. Before the policy update request is sent to the policy server, the request is processed in the CCU to fulfill the desired security properties, as specified above. First, the CCU create a message and request for trusted timestamping issued by HSM. This trusted time stamping proves the existence of certain data before a certain point without the possibility that the ECU can backdate the timestamps. In addition, the message(`m`) includes a random number to prevent replay attacks.

   Next, a `Sig(SK, m)` is created on `m` using `Pseudonym SK` of CCU. Now, the entire `m` is integrity-protected as shown in step 1. The Pseudonym keys are used in order to protect the driver's privacy, the vehicle should not be traceable over a long time or external systems. Pseudonyms needs certificates as do long-term identifiers to declare them valid. We follow the same approach of using Pseudonym certificates for all V2X as specified in [14]. A Pseudonym certificate is also sent along with the message for signature verification.

```
    Sig(SK_ccu,(Request_policy_latest_versions,time_stamp, rand)),
        Cert_ccu(SK_ca, Pseudo_PK_ccu)     (1)
```

2. *Policy List Update.*On policy server side, the following procedure occurs, as illustrated by the sequence diagram in Figure 9. The first certificate is verified using `Ver_cert`, and at the same time signatures are verified using `Ver_sig(PK, Sig_data, m)`, allowing the policy server to verify the origin and integrity of a given signed message. The process of `Ver_sig(PK,Sig_data, m)` (as shown in step 2) is proposed to ascertain whether a given message has been signed by the private key that corresponds to a given public key, retrieved from the certificate.

```
    Pseudo_PK_ccu:= Ver_cert(Cert_ccu(SK_ca,Pseudo_PK_ccu))
    Boolean:= Ver_sig(Pseudo_PK_ccu, Sig_data,
        Request_policy_latest_version, time_stamp, rand))     (2)
```

   If the two values are identical, the verification is successful and proves that the message has been signed with the private key that corresponds to the public key used in the verification process, proving that the time stamp and message is unaltered and was issued by the CCU. If the two values differ from one another, this means that the digital signature is invalid and the verification is unsuccessful. The policy server generate a list (`Latest_policy_versions`) of all the critical and complimentary updates released for the on-board system. Next, a trusted time stamp and random number is included with the message `m` and `Sig(SK, m)` is generated on `m` using `SK` of Manufacturer. This signed message along with signed certificate is sent to the vehicle as shown in step 3.

```
    Sig(SK_ps, (Latest_policy_versions, time_stamp, rand)), Cert_ps(
        SK_ca, PK_ps)     (3)
```

3. *Policy List Comparison.* This received message includes all information that is necessary to deliver this message to the vehicle (i.e., CCU). In addition, the message includes a list of `latest_policy_version, trusted time stamp, rand` and `PK` of the policy server and signed certificate of the policy server. In order to ensure that the list is issued by the policy server on a specific time duration and unaltered on the air, CCU will verify the certificate and signatures on the `m` as shown in step 4 of the received message.

```
    Pseudo_PK_ccu:= Ver_cert(Cert_ps(SK_ca, PK_ps))
    Boolean:= Ver_sig(PK_ps, Sig_data, (Latest_policy_versions,
        time_stamp, rand))
    compare_policy_versions (Latest_policy_versions,
        previously_stored_policy_versions)    (4)
```

   If the two values are identical, the `compare_policy_versions` function then compares this list with the list of installed updates in the vehicle system, and displayed a message on the Human-Machine Interface (HMI) to the user informing them of new critical/complimentary updates if they were available. If the user accepts the new policy updates, CCU will send a *Sig* message to the Policy server for establishing a secure, authentic and confidential channel between policy server and an appropriate ECU_{PDM}, which requires policy update. This message includes an encrypted certificate of the intended `ECU`, this certificate may also contain certain attributes of the certified unit (ECU), such as type, functions, allowed applications etc. In addition, this message includes `trusted time stamp` and `random number` as shown in step 5.

```
    Sig(SK_ccu, (Encrypt (PK_ps , Cert_ecu(SK_mv, PK_ecu))),time_stamp
        , rand), Cert_ccu(SK_ca, Pseudo_PK_ccu)    (5)
```

4. *Generation of Session Key Pair.* In the policy server side, same steps are again performed to ensure the integrity, freshness and origin authenticity by using `Ver_sig(SK, Sig_data, m)`. If the two values are identical, the policy will generate a session key pair (Key pair also for symmetric EVITA keys, which feature complementary functions sign/verify and encrypt/decrypt). Following HSM approach, where multiple key-properties may be set, only policy server may sign and encrypt policies, whereas receiving ECU can verify and decrypt policies using he same key material. The generated Key `Ks` stays at the policy server and verification key `Ks'` is exported to other `ECU`. Next, the policy server will export the generated session key as shown in step 6, and encrypt with `PK_ecu`, which can be retrieved from certificate.

```
    Pseudo_PK_ccu:= Ver_Cer(Cert_ccu(SK_ca, Pseudo_PK_ccu))
    Boolean:= Ver_sig(Pseudo_PK_ccu, Sig_data, (Encrypt (PK_ps ,
        Cert_ecu(SK_mv, PK_ecu1)), time_stamp, rand))
    Decrypt(SK_ps, Cert_ecu1(SK_mv, PK_ecu))
    SesK-Handle:= create_random_key(use_flags= verify|decrypt})
    Export_Sesk:= key_export(transport_key_handle = <PK-ECU-Handle>,
        kh = <Sesk-Handle>))        (6)
```

   This exported key is then *Sig* together with `trusted time stamp` and `rand` and delivered to the vehicle as shown in step 7.

```
Sig(SK_ps, (Export_Sesk, time_stamp, rand))   (7)
```

5. *Key Import.* CCU verifies the signature of the received message as shown in step 8. If this is true, CCU `Sig(SK, m)` the message and transmits message to the correct ECU. This message includes all information that is necessary to deliver this message to the correct ECU. ECU validates based on the policy whether CCU is allowed to deliver message or not.

```
Boolean:= Verify_sig(SK_ccu, Sig_data, (Exported_Sesk, time_stamp,
    rand))
key_import(transport_key_handle =<SK_ecu-Handle>, import-Key=
   Export_Sesk)  (8)
```

   ECU verifies whether `m` is authentic and fresh by verifying the signature `Ver_sig(SK, Sig_data, m)` and validating whether the integrated random number has not already been used in the past. If the two values are identical, ECU will import the session key `Ks'` as shown in step 8. If the key is successfully imported, the ECU then transmits an `Acknowledgment` message to the policy server via CCU as shown in step 9.

```
Sig(SK_ecu, (Ack, time_stamp, rand))
Boolean:= Ver_sig(SK_ecu, (Ack, time_stamp, rand))
Sig(SK_ccu, (Ack, time_stamp, rand)), Cert_ccu(SK_ca,
   Pseudo_PK_ccu)    (9)
```

6. *Policy Upload.* Before the new security policies are sent to the vehicle, the security policies are processed at the `Policy Server` side. First, the new security policy is transformed into binary, as specified in [64] and divided into n data fragments `(D1 to Dn)`, depending on the size of the security policy. Next, a `Gen_MAC(Ks, m)` method is used to generate the MAC. This message includes `trusted time stamp`, `security policy` and `random number`. The message is then delivered to the on-board `ECU` as shown in step 10. Furthermore, it is reasonable to consider that some security policies are confidential and need to be protected, in such case security policies are encrypted with `Ks`.

```
MAC(Sesk, (Security_policy, timestamp, rand)) (10)
```

7. *Secure storage of security policies.* The received message (via CCU) is verified `Ver_MAC(Ks', m, Mac_data)` and compared with the received MAC. If the two values are identical, this proves that the message is fresh, has been sent by authentic source and not modified in the air. If the two values differ from one another, this means that the MAC is invalid and the verification is unsuccessful.

   The security policy is downloaded block by block and stored in the memory using the Secure Storage module, as specified in [64]. The new/updated security policy will be configured during next secure bootstrapping process.

3.4.3 Policy Configuration Protocol

This section provides an overview of the security policy configuration protocol. We are considering two possible scenarios for policy configuration: policy configuration during secure bootstrapping process and runtime policy configuration (more specifically after intrusion detection). During the secure bootstrapping process, ECU checks whether an external programming request (new/updated policy set) has been received. The information whether an external programming request has been received or not shall be stored in non-volatile memory, e.g. Electrically Erasable Programmable Read-Only Memory (EEPROM). After this check has been done, the policy loader shall reset the variable, forcing the ECU into programming mode. To start the policy loader the needed software modules (i.e., PDM, CCM, etc.) shall be initialized. We are following a similar approach to configure security policies as specified in HIS Flashloader Specification [46] for installing new firmware. However, in policy loader process only new uploaded security policies are retrieved from Secure Storage Module (SSM) and PDM(s)/Policy Enforcement Point (PEP) are initialized with the new security policy as shown in Figure 10. SSM checks whether the requested operation is allowed or not (file access control) using `PEP_check` method. If allowed, it uses a file system plug-in to open the file in the protected file system. It may be necessary to have the whole file to be verified for the selected security requirements once on open, which would be the best choice from the security point of view. The storage plug-in reads blocks from the storage and enforces confidentiality, integrity/authenticity, and freshness as well as the block-order at block level using the block protection layer as specified in [64]. Next, the PDM will set the new security policy and provides all authorization functionality based on new set security policy as shown in step 1.

```
load_security_policy(Security_policy)
PDM_set_security_policy(Security_policy) (1)
```

In the second scenario, the SWD sends a message `PDM_set_security_policy` (see Figure 11) to the PDM (as shown in step 2 to set/restrict or reconfigure possible security policy parameters, after detecting intrusion). Actions may consist in updating the contextual information available at PDMs, i.e. updating the state of the PDM, or even updating the policy of PDMs (filtering or endpoint access control), i.e., reconfiguring its access control rules. The suggested protocol is illustrated in Figure 10.

```
Sig (SK_ecu, (PDM_set_security_policy, time_stamp, rand)) (2)
```

The received message is validated based on the policy whether ECU_{swd} is allowed to deliver message to ECU_{pdm}. If this is true, ECU_{pdm} verify (as shown in step 3) whether `m` is authentic and fresh by verifying the signature `Ver_sig(SK,Sig_data, m)`. Next, PDM will restrict the security rules and parameters, depending on the intrusion or PDM will send a request to Secure Storage Module SSM to load new/reconfigure security policy.

```
isAuthentic:=EAM_verify_authentication_ticket(Cer_ecu(SK_mk,PK_ecu))
PEP_check(Cer_ecu(SK_mk,PK_swd-ecu), operation, object)
Boolean:= Ver_sig(PK_ecu, Sig_data, (PDM_set_security_policy, time_stamp,
    rand))
SSM_read(Security_policy_identifier) (3)
```

SSM checks whether the requested operation is allowed or not (file access control) using `PEP_check` method and sends a response to the PDM. Last, a security policy is set `PDM_set_secutity_policy` and an acknowledgment is sent to the ECU_{swd} (see step 4).

```
PDM_set_security_policy(Security_policy)
Sig(SK_ecu, (ACK, time_stamp,rand)) (4)
```

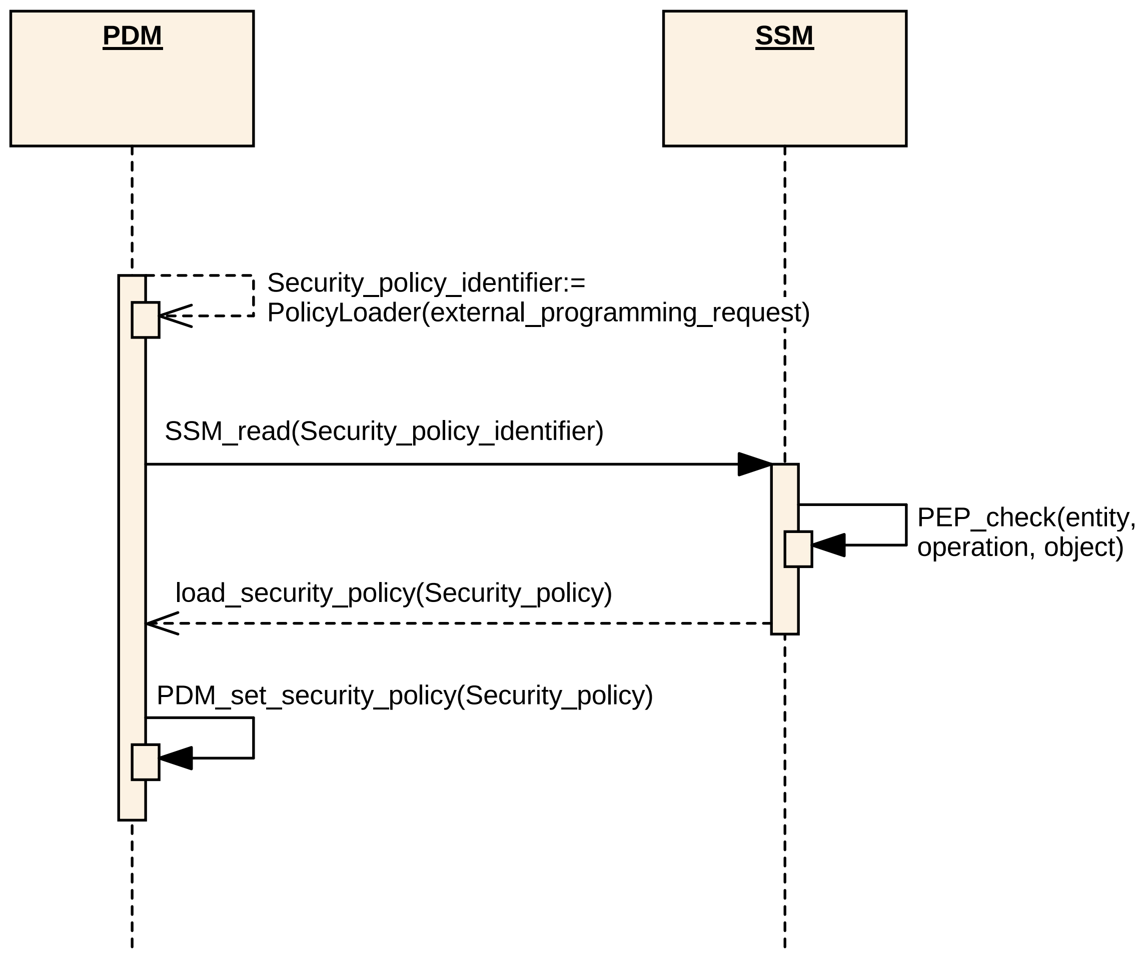

PDM
SSM
Security_policy_identifier:=
PolicyLoader(external_programming_request)
SSM_read(Security_policy_identifier)
PEP_check(entity,
operation, object)
load_security_policy(Security_policy)
PDM_set_security_policy(Security_policy)

Figure 10 Policy Configuration Protocol – Secure Bootstrapping

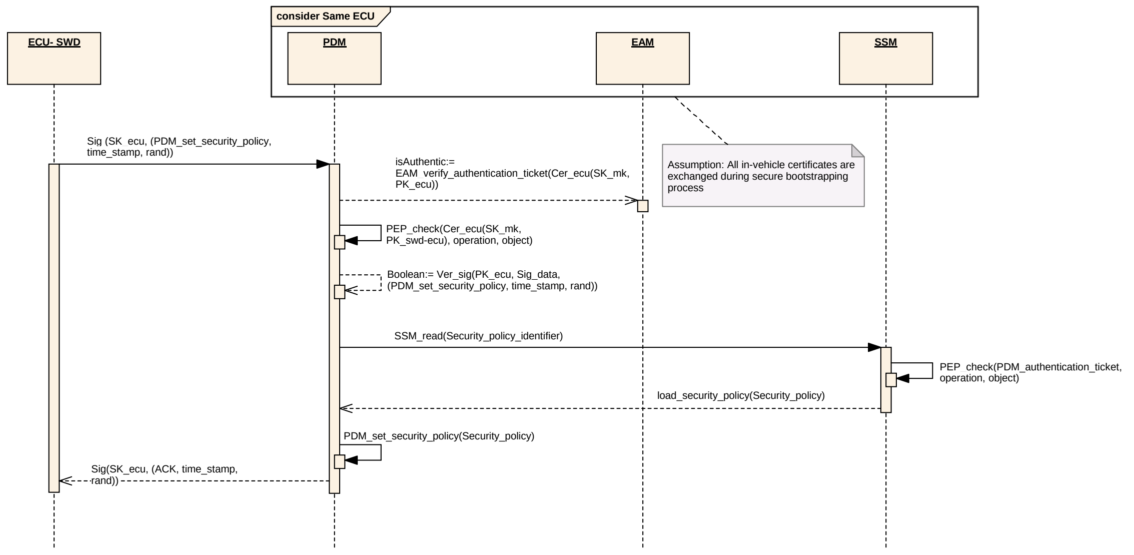
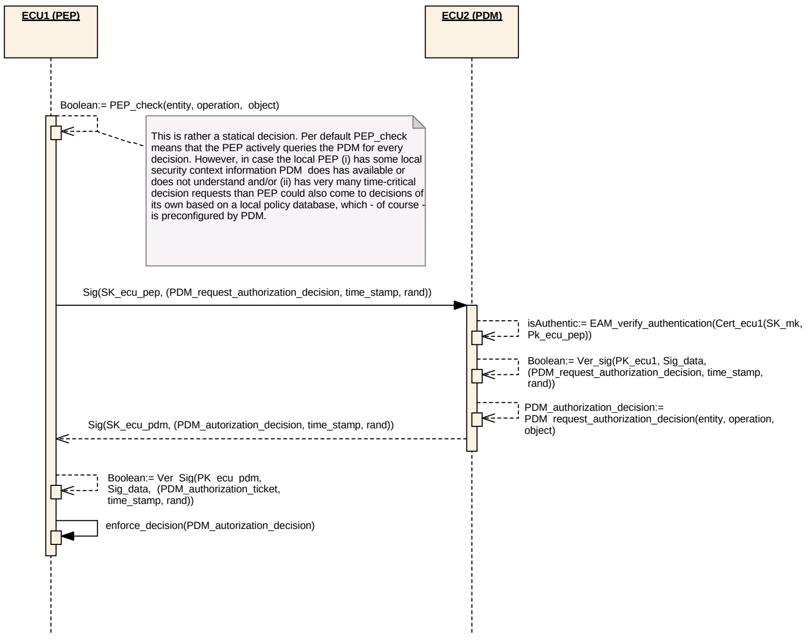

consider Same ECU
ECU- SWD
PDM
EAM
SSM
Sig (SK_ecu, (PDM_set_security_policy, time_stamp, rand))
isAuthentic:= EAM_verify_authentication_ticket(Cer_ecu(SK_mk, PK_ecu))
Assumption: All in-vehicle certificates are exchanged during secure bootstrapping process
PEP_check(Cer_ecu(SK_mk, PK_swd-ecu), operation, object)
Boolean:= Ver_sig(PK_ecu, Sig_data, (PDM_set_security_policy, time_stamp, rand))
SSM_read(Security_policy_identifier)
PEP_check(PDM_authentication_ticket, operation, object)
load_security_policy(Security_policy)
PDM_set_security_policy(Security_policy)
Sig(SK_ecu, (ACK, time_stamp, rand))

Figure 11 Policy Configuration Protocol – SWD – runtime policy configuration

Restricted access control and reconfiguration of security policy allow the behavior of the old policy while potentially reducing the risk measure. This is particularly useful within such infrastructures as it enables us to: detect, analyze and resolve multiple conflicts; decide if a change in the environment necessitates a security reconfiguration and to decide if a suitable level of security interoperability between heterogeneous systems is achievable.

3.4.4 Policy Enforcement Protocol

1. **Local Policy Decision:** This section provides an overview of the policy enforcement protocol in order to enforce security policies on executions and communications requests in each ECU. A security policy is a flexible set of security configuration, authorization tickets and trust statements. The set of rules to apply in order to reach decisions policies, the process of applying the rules `Policy Enforcement`, and the checkpoints themselves – PEP. A `SecurityConfigurationTicket` is a special set of rules to configure a certain PEP. Thus, security configuration tickets usually are, after their successful verification using the integrated security credential, immediately applied by PDM to the corresponding PEP. Each PEP defines check methods. The semantic and number of these methods depends on the nature of the access checks to be made by the PEP. This is rather a statical decision. Per default `PEP_check` means that the PEP actively queries the PDM for every decision. However, in case


ECU1 (PEP)
ECU2 (PDM)
Boolean:= PEP_check(entity, operation, object)
Sig(SK_ecu_pep, (PDM_request_authorization_decision, time_stamp, rand))
isAuthentic:= EAM_verify_authentication(Cert_ecu1(SK_mk, Pk_ecu_pep))
Boolean:= Ver_sig(PK_ecu1, Sig_data, (PDM_request_authorization_decision, time_stamp, rand))
PDM_authorization_decision:= PDM_request_authorization_decision(entity, operation, object)
Sig(SK_ecu_pdm, (PDM_autorization_decision, time_stamp, rand))
Boolean:= Ver_Sig(PK_ecu_pdm, Sig_data, (PDM_authorization_ticket, time_stamp, rand))
enforce_decision(PDM_autorization_decision)

This is rather a statical decision. Per default PEP_check means that the PEP actively queries the PDM for every decision. However, in case the local PEP (i) has some local security context information PDM does has available or does not understand and/or (ii) has very many time-critical decision requests than PEP could also come to decisions of its own based on a local policy database, which - of course - is preconfigured by PDM.

Figure 12 Policy Enforcement Protocol

the local PEP *(i)* has some local security context information PDM has available or does not understand and/or *(ii)* has very many time-critical decision requests than PEP could also come to decisions of its own based on a local policy database. The suggested protocol is illustrated in Figure 12.

- *Autonomous PEP.* PEP acts as ancillary PDM based on local security policy decision database and security context information (general and/or promptly only) available at the respective security module or security program implementing a PEP (e.g., secure display). The authorization for local autonomous decisions and the security policy database is configured by PDM and checked at locally as shown in step 1 if PEP does not understand or is not able to take decision based on locally configured security policy, PEP will forward the request to respective PDM.

```
        PEP_check(entity, operation, object)   (1)
```

- *Forward PEP.* PEP simply acts as decision forwarder and hence forwards any decision requests to PDM and returns PDMs answer to the local caller. This is the PEP default implementation, which may have some temporal and/or contextual limitations. Before the request is sent to the ECU_{PDM}, the request is processed at the ECU_{PEP} to fulfill the desired security properties. The request `PDM_request_authorization_decision` with `trusted time stamp` is

Sig(SK,m) and sent to ECU_{PDM}. In addition, the message includes a random number to prevent replay attacks. It is thus of the following form:

```
        Sig(SK_ecu_pep, (PDM_request_authorization_decision,
            time_stamp, rand)) (2)
```

In case of ECU_{PEP} with $light_{\text{HSM}}$, ECU_{PEP} generates a MAC (Gen_MAC(K_mac, m)) and sends it to the ECU_{PDM}. At ECU_{PDM} side, the following procedure occurs, as illustrated in step 3. First ECU_{PEP} authentication is verified using EAM_verify_authentication and at the same time signatures are verified using Ver_sig(PK,Sig_data, m), allows ECU_{PDM} of given signed message to verify its real origin and its integrity. The process of Ver_sig(PK,Sig_data, m) (as shown in step 2) is purposed to ascertain if a given message has been signed by the private key that corresponds to a given public key, retrieved from the already received certificate (during secure bootstrapping process).

```
        isAuthentic:= EAM_verify_authentication(Cert_ecu1(SK_mk,
            Pk_ecu_pep))
        Boolean:= Ver_sig(PK_ecu_pep, Sig_data, (
            PDM_request_authorization_decision, time_stamp, rand))
              (3)
```

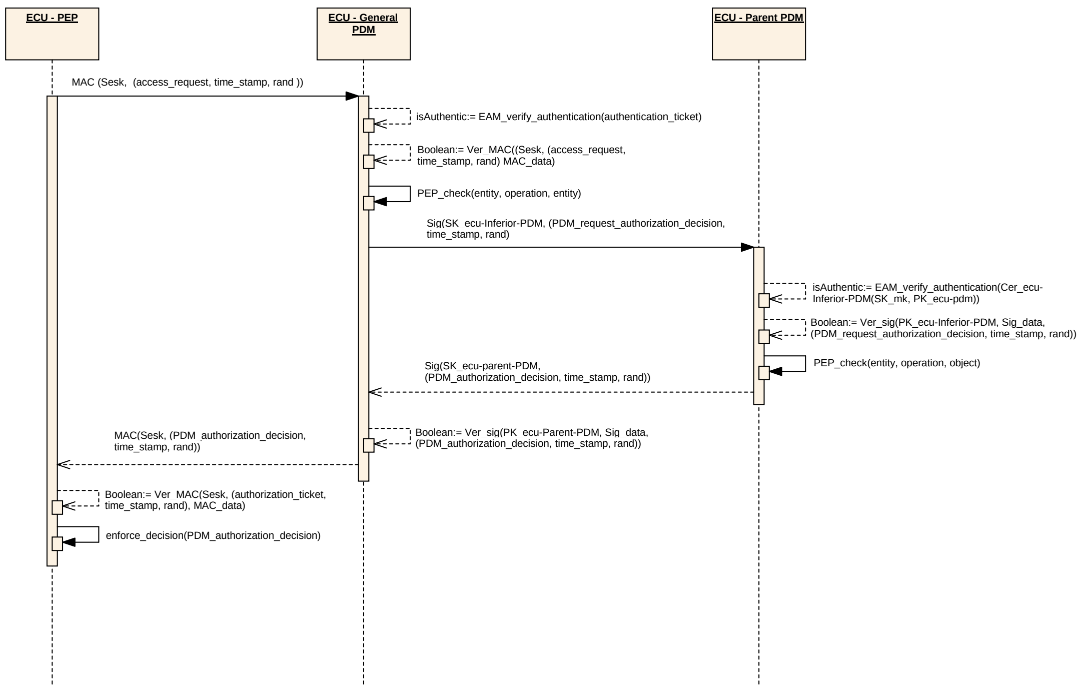
If the two values are equal to true, the verification is successful and proves that the message has been signed with the private key that corresponds to the public key used in the verification process, proving that the time stamp and message is unaltered and was issued by the ECU_{PEP}. If the two values differ from one another, this means that the digital signature is invalid and the verification is unsuccessful. The ECU_{PDM} generate s PDM_authorization_decision. Next, a trusted time stamp and random number is included with the message m, and the signature (Sig(SK, m)) is generated. This signed message is sent to the ECU_{PEP} as shown in step 4.

```
        Sig(SK_ecu_pdm, (PDM_autorization_decision, time_stamp,
            rand))    (4)
```

Last, message is verified as shown in 5, and ECU_{PEP} will enforce the access decision. This collaboration provides mechanisms to ensure that authorizations are granted in a consistent and efficient manner throughout the system.

```
        Boolean:= Ver_Sig(PK_ecu_pdm,Sig_data, (
            PDM_authorization_ticket, time_stamp, rand))
        enforce_decision(PDM_autorization_decision)    (5)
```

2. **Distributed Policy Decision:** Security policies are the basic for all authorizations within the vehicular system. Hence, mechanisms for creating, changing, and managing policies are very important. A very important functionality needed to operate in such architectures is distributed authorization. Next, the understanding of distributed authorization in the context of on-board architecture is explained in more detail. In the case of authorization decisions in heterogeneous systems, policies offer the potential to abstract away from the details of who is allowed to access



ECU - PEP
ECU - General PDM
ECU - Parent PDM
MAC (Sesk, (access_request, time_stamp, rand))
isAuthentic:= EAM_verify_authentication(authentication_ticket)
Boolean:= Ver_MAC((Sesk, (access_request, time_stamp, rand) MAC_data)
PEP_check(entity, operation, entity)
Sig(SK_ecu-Inferior-PDM, (PDM_request_authorization_decision, time_stamp, rand)
isAuthentic:= EAM_verify_authentication(Cer_ecu-Inferior-PDM(SK_mk, PK_ecu-pdm))
Boolean:= Ver_sig(PK_ecu-Inferior-PDM, Sig_data, (PDM_request_authorization_decision, time_stamp, rand))
PEP_check(entity, operation, object)
Sig(SK_ecu-parent-PDM, (PDM_authorization_decision, time_stamp, rand))
Boolean:= Ver_sig(PK_ecu-Parent-PDM, Sig_data, (PDM_authorization_decision, time_stamp, rand))
MAC(Sesk, (PDM_authorization_decision, time_stamp, rand))
Boolean:= Ver_MAC(Sesk, (authorization_ticket, time_stamp, rand), MAC_data)
enforce_decision(PDM_authorization_decision)

Figure 13 Distributed Policy Decision Protocol

which services, under which conditions. In a vehicular authorization environment, however, there are more entities in the system, which can be both authorizers and requesters, and probably are unknown to each other. To simplify authorization in on-board architecture, we need a system in which access-control decisions are based on authenticated attributes of the subjects, and attribute authority is decentralized, in order to attain more fine grain access rights. The suggested protocol illustrates the procedure of requesting distributed access control decision. In case a certain PDM cannot decide a certain request he does not understand or simply not authorize to decide on certain issue, he may forward the request to other parent PDMs. The suggested protocol is illustrated in Figure 13.

- *General PDM.* A PDM usually acts autonomously in its domain where he is assigned. He makes decisions on all authorization requests he understands, which means, on all queries he has the respective policy interpreter, policy decision engine, and corresponding policies available. Note, for interacting with other PDMs or PEPs all modules have to establish and verify their trust relationship and communicate with each other at least using an authentic channel (e.g., as provided by CCM).
- *Inferior PDM.* As stated before, a PDM can pre-configure some autonomous PEPs to partly act on its own. This is the case where for instance the general

PDM has necessary (application-specific) context information not efficiently available (e.g., secure GUI window manager PEP).

- *Parent PDM.* In case a PDM cannot decide a certain request since he does not understand the policy language, has corresponding policies not available or is simply not authorized to decide on certain issues, he may have a parent PDM or even parent PDMs configured to forward the request and to receive the respective decision.

3.5 Secure Bootstrapping Protocol

3.5.1 Bootstrapping Components

- **Policy Propagation :** Our high-level goal is to auto-configure security components and establish trust during vehicle bootstrapping. To achieve this goal, we must assume that the vehicular on-board network already possesses some trusted entities such as a Trusted Policy Server (TPS), HSM, or KeyMaster. Specifically, we make two initial trust assumptions. First, we assume the bootstrapping policies are stored in the secure data storage, and communicate with other modules using CCM secure channel. This TPS is also assumed to be trusted due to the limited interaction, so it can be only modified using Metameta access policies (OEMs). We also assume HSM as a trusted entity. Without this assumption, it is difficult to enable secure storage of keys and computation of cryptographic operations.
- Secure Channels
- Key Management and initialization
- Time Synchronization
- Platform Integrity Checks

3.5.2 Time Synchronization Protocol

Time synchronization is performed in a two-step approach. In step 1, one specific ECU in the vehicle obtains the time from an external source and guards it in its locally protected real-time clock. In step 2, synchronization of the individual ECUs is performed, so that the HSM's tick-counters represent the actual time.

Step 1 Time synchronization with the backend server can be performed using standard protocols, that are secure appropriately. The Global Positioning System (GPS) signal, that is usually taken as local reference, does not offer advanced security as no cryptographic measures are taken. This means that the radio signal can be forged by attackers. An existing IP network connection can be used to securely obtain time from Network Time Protocol (NTP) services capable of Secure Sockets Layer (SSL)/Transport Layer Security (TLS). To achieve a higher level of accuracy, multiple servers and multiple measurements shall be taken, as well as checked for plausibility in combination with data acquired from a different source (e.g. GPS).

Step 2 Time synchronization may be performed between any two ECUs whenever one of them is equipped with a secure Real Time Clock (RTC). The ECU with the RTC signs the actual Coordinated Universal Time (UTC) value together with a random number recently obtained from the other one. The signature is then sent back to and verified by the other ECU. The time synchronization protocol is shown in Figure 14. There exist other similar time synchronization protocols which are even more precise, for example by introducing multiple measurements. However, they are impractical for our purpose since they would cause too much overhead regarding the necessary infrastructure and communication.

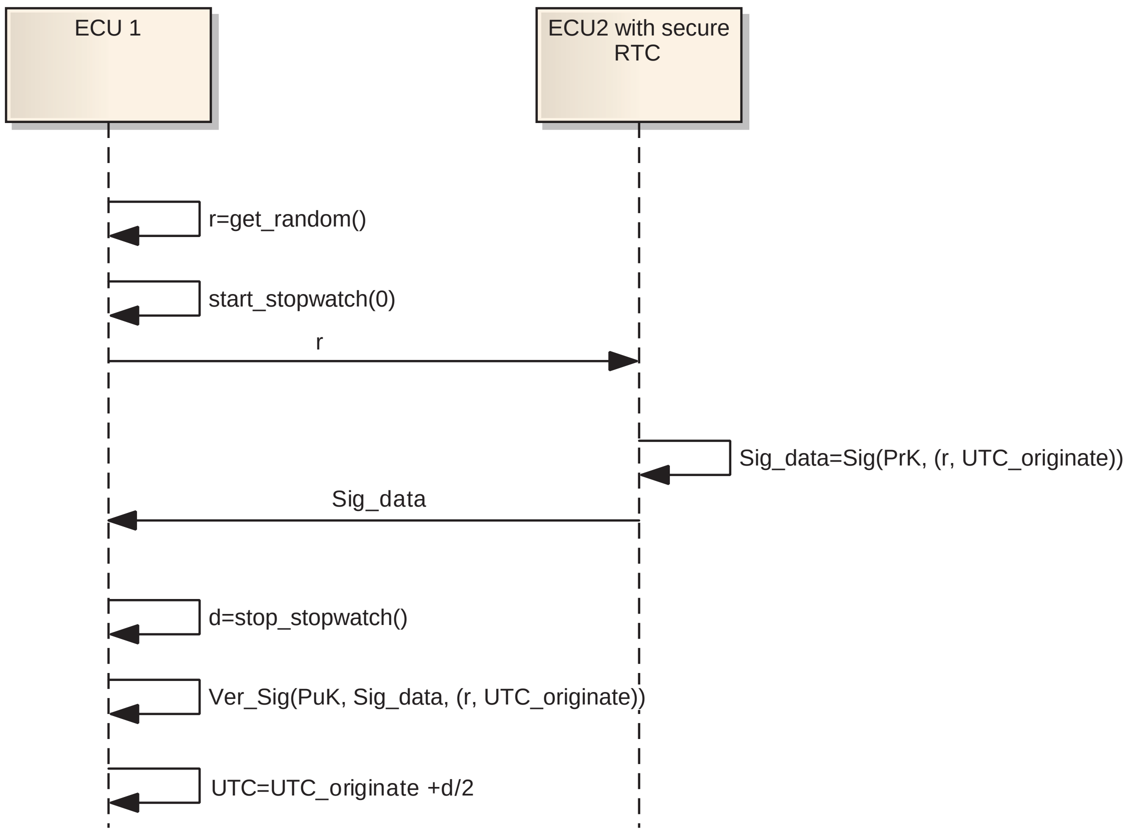


Figure 14 Time Synchronization Protocol between ECUs, one of them equipped with a secure RTC.

3.5.3 Platform Integrity and On-board Network Integrity

We assume that the flash memory is part of the system-on-chip platform as described in [64]. Program code (i.e., the firmware) is enhanced with an authentication code that is validated upon boot.

If the flash memory is outside of the on-chip platform (i.e., outside the HSM), its contents must not only maintain integrity but also need to be encrypted to satisfy the confidentiality requirement from [55].

On-Board Network Integrity In Figure 15 we show how the KeyMaster node of a domain can verify that ECUs have not been replaced, i.e., that a specific node does still have access to pre-shared key material. Requests to respond are sent to all paired nodes. Nodes reply with the requested challenge and a MAC based on their PSK.

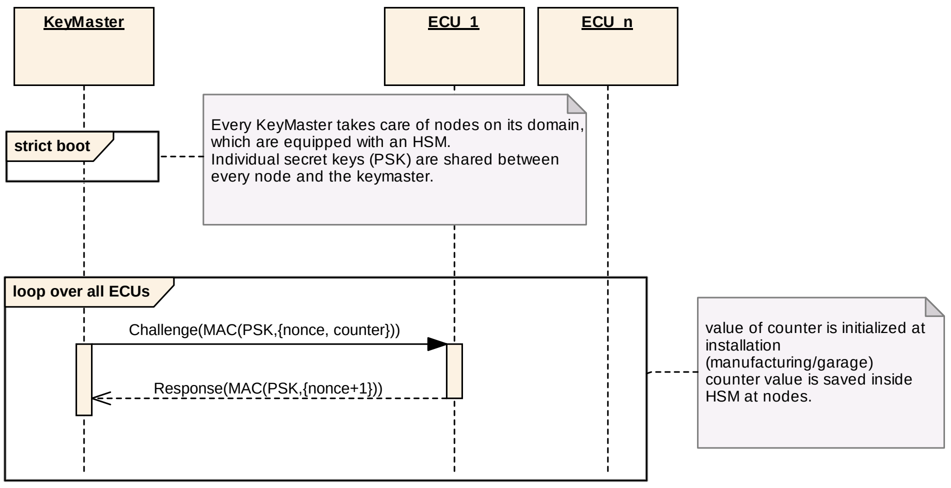

KeyMaster
ECU 1
ECU n
strict boot
Every KeyMaster takes care of nodes on its domain, which are equipped with an HSM. Individual secret keys (PSK) are shared between every node and the keymaster.
loop over all ECUs
Challenge(MAC(PSK,{nonce, counter}))
Response(MAC(PSK,{nonce+1}))
value of counter is initialized at installation (manufacturing/garage) counter value is saved inside HSM at nodes.

Figure 15 On Board Network Integrity Protocol: Keymasters require all nodes of their domain to respond with challenge-response according to a PSK.

3.6 Secure over-the-air firmware updates

Current research activities in vehicular on-board IT architectures basically follow two key trends: unification of network communication and centralization of functionality. Recent on-board IT architectures comprise a very heterogeneous landscape of communication network technologies, e.g., CAN, LIN, FlexRay and MOST. IP-based communication is currently being researched as a technology for unifying the overall interconnection of ECUs in future on-board communication systems [53]. In addition, there is a shift towards multipurpose ECUs and usage of flash memory technology in the microcontrollers. Besides these trends in the design of automotive on-board IT architectures, new external communication interfaces, fixed and wireless are becoming an integral part of on-board architectures. One key factor for this development is the integration of future e-Safety applications based on V2X[6] communications that have been identified as one promising measure for increasing the efficiency and quality of operational performance of all vehicles and corresponding intelligent transportation systems.

Firmware updates are crucial for the automotive domain, in which recalls are a very costly activity and thus should be avoided where possible. The practicability of remotely updating devices has been shown by Google for their Android telephones. With this,

[6]V2X is an abbreviation that stands for any external vehicular communications such as Vehicle-to-Vehicle (V2V) or Vehicle-to-Infrastructure (V2I) communications.

they have a powerful tool to immediately react on discovered security flaws in very short time [60]. In the automotive domain, update intervals are calculated in quarters of a year and not quarters of a day right now. This paradigm is about to change and security mechanisms within the car provide the necessary building blocks. With the arising "always-connected" infrastructure, it will be possible to perform over-the-air (OTA) diagnosis and OTA firmware updates (see Fig. 16), for example. This will provide several advantages over hardwired access, such as saving time by faster firmware updates, which improves the efficiency of the system by installing firmware updates as soon as they are released by the car manufacturer. However, adding new in-vehicle services not only facilitates novel applications, but also imposes stringent requirements on security, performance, reliability, and flexibility. As discussed in [36], in-vehicle components need not only to be extremely reliable and defect free, but also resistant to the exploitation of vulnerabilities. Although on-board bus systems are not physically accessible (apart from via diagnostic interfaces), this provides only a limited degree of security for vehicles that are in wireless communication with other vehicles and devices (e.g., consumer devices connected to the vehicle).

As seen in [36, 55], attacks on the in-vehicle network have serious consequences for the driver. If an attacker can install malicious firmware, he can virtually control the functionality of the vehicle and perform arbitrary actions on the in-vehicle network. Furthermore, since the ECU itself is an untrusted environment, there exist challenges in how to securely perform cryptographic operations (i.e., encryption/decryption, key storage). Thus, it does not make much sense if the verifier software runs from the same flash as the software to be verified. In the following we present a generic firmware update protocol that can be used for both, hardwired and remote firmware flashing. The protocol has especially been designed with respect to the above mentioned functional and non-functional requirements.

3.6.1 Over-the-air update requirements

Before sketching the protocol, we describe constraints/requirements responsible for secure firmware updates, including the secure storage of key material in the Diagnostic Tool (DT) and secure transport of firmware data over the on-board bus system.

The most desired security properties in order to provide secure firmware updates are *Authentication, Integrity, Freshness*, and *Confidentiality*, as identified in [55]. The specified protocol provides means to satisfy these security requirements. The protocol shows how modules of the HSM are used and how they interact in order to ensure secure firmware is downloaded and installed securely in the vehicle.

Hardware Security Module in the Diagnostic Tool: As stated in [55], there are numerous scenarios, where an attacker attacks on the DT such as, the attacker injects bogus authority keys into the ECU, through DT, which compromise the overall security of the vehicular on-board architecture. In particular, this means that the DT stores challenges and public strings for key recovery (i.e., ECU unlock key) and is therefore responsible for the security of the subsystem. Therefore, this information needs to be stored securely on the DT-side. An additional advantage of HSM is the resistance against physical tampering of the DT. Any damage to the HSM changes the behavior and, therefore, prevents the extraction of secret key material.

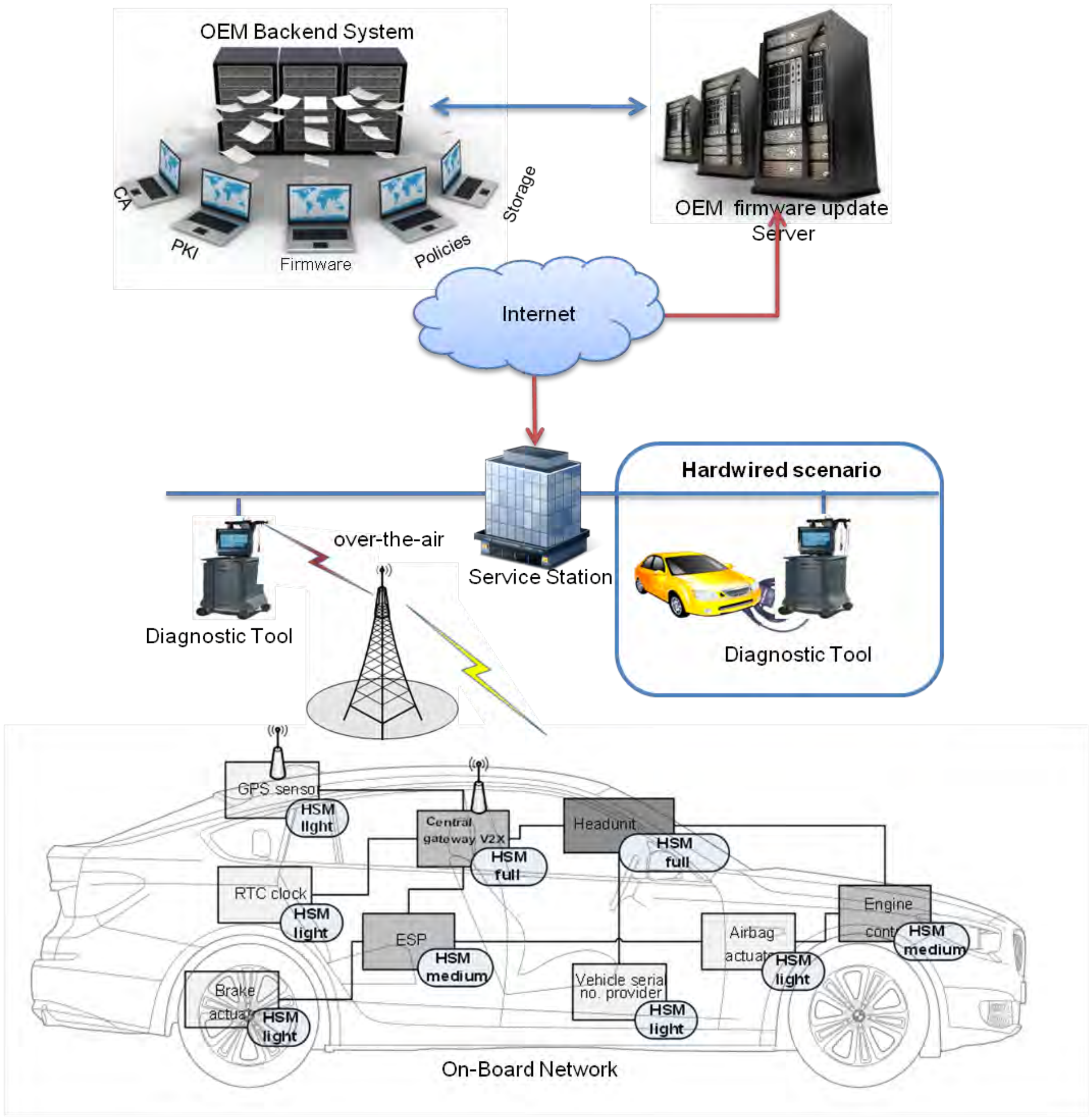


Figure 16 Over-the-Air Firmware Updates

3.6.2 Protocol description

Remote Diagnosis: In the over-the-air firmware update scenario, a service station using a diagnostic tool (DT) connect remotely, using V2I (Vehicle-to-Infrastructure) infrastructure, to assess the state of the vehicle (see Figure 16). To know which version is installed, a diagnosis of the vehicle is required to have all necessary information such as ECU type, firmware version, and date of last update. An employee of the station using the diagnosis tool establish a secure connection with the vehicle, at the ECU level, in order to determine the current state of the vehicle. To do so, DT creates a session key M_{sk} (exportable), by sending a HSM command `Create_Random_key` and specifies set of allowed key properties such as, `use_flag = sign|verify, encrypt|decrypt`. It then `export` the M_k using P_k_ccu (Public key of the CCU) as a transport key (T_k) and transmit it to the vehicle. Here, the central communication unit is the first receiving entity in the vehicle, responsible for receiving and distributing V2X messages to the in-vehicle network.

In the vehicle, the CCU, equipped with the Full HSM module and acting as a KM node, receives the connection request. The authorization for the connection is verified in the CCU. The message $\overrightarrow{m}$ is checked for freshness, integrity and the service station is authenticated. If the check succeed, CCU-KM import the key in the HSM. It then export the received $Exported\_M_k$ with the corresponding P_k_ecu or Psk, depending on the ECU type, and distribute to the target ECU in order to enable end-to-end communication. This message includes all information that is necessary to deliver this message to the correct ECU. On the ECU side, ECU verifies integrity, authenticity and authorization of CCU/DT based on the policy whether DT is allowed to deliver a message or not. If this is true, and message is fresh, ECU import the M_k in the HSM. Once M_k key have been imported, acknowledgment is sent back to the DT (see Algorithm 1). After this acknowledgment frame, the DT sends, depending on the option chosen by the employee of the service station, requests to read out diagnosis information (State/Log information) from the ECU it wants to check.

Algorithm 1 Remote Diagnosis

```
Require: Signature verification Key $V_k$ of DT, CCU, ECU are pre-loaded
Ensure: Establishing a fresh and authentic session between DT and ECU based on a symmetric session key $M_k$, where
    CCU-KM acts as a Key Master Node.

    1. DT $\rightleftarrows$ HSM: $M_k$-handle:= $create\_random\_key$ ($use\_flag$ = sign| verify, encrypt| decrypt, export, $use\_authorization\_data$)
       DAT: $Exported\_M_k$ := $key\_export$( $T_k$-handle = $\lessdot$ $P_k\_ccu$-handle $\gtrdot$, kh=$\lessdot$ $M_k$-handle$\gtrdot$)
       DT $\rightarrow$ CCU-KM: $\left\{(Exported\_M_k, Ts)\,,\ \{\widehat{\sigma}\}_{S_k\_dt}\right\}$

    2. CCU $\rightleftarrows$ HSM: $M_k$-handle := $key\_import$ ($T_k$-handle = $\lessdot$ $S_k$\_ccu-handle $\gtrdot$, kh= $\lessdot$ $M_k$-handle $\gtrdot$)
       CCU-KM: $Exported\_M_k$ := $key\_export$($T_k$-handle = $\lessdot$ $P_k$\_ecu-handle | Psk-handle$\gtrdot$, kh=$\lessdot$ $M_k$-handle$\gtrdot$)
       CCU-KM $\rightarrow$ ECU: $\left\{(Exported\_M_k, Ts)\,,\ \{\widehat{\sigma}\}_{S_k\_ccu}\right\}$

    3. ECU $\rightleftarrows$ HSM: $M_k$-handle := $key\_import$ ($T_k$-handle = $\lessdot$ $S_k$\_ecu-handle| Psk-handle $\gtrdot$, kh= $\lessdot$ $M_k$-handle $\gtrdot$)
       CCU-KM $\leftarrow$ ECU: $\left\{(ACK, Ts)\,,\ \{\widehat{\sigma}\}_{S_k\_ecu}\right\}$

    4. DT $\leftarrow$ CCU-KM: $\left\{(ACK, Ts)\,,\ \{\widehat{\sigma}\}_{S_k\_ccu}\right\}$
```

Discussion – Advance Notification: Due to legal reasons and to allow for flexible deployment, we consider that service station sends an advance notification of possible firmware updates, if the type is the expected one. This advance notification is intended to help customers plan for effective deployment of updates, and includes information about the number of new updates being released. These updates still need to be `Approved` for install before downloading. The customer receives this information on the vehicle HMI and can decided about possible deployment (i.e., `Install, Decline, Decide later`). Only updates that have the approval status `Install` will be downloaded to the vehicle. If firmware update request is approved for install, it requires that vehicle is stopped and have access to the V2I infrastructure for receiving firmware data. Disabling ECU while vehicle is running may cause some serious problem, depending on the function ECU is responsible for. Thus, we consider some additional checks, provided by the on-board system, to ensure that vehicle is stopped and have access to the infrastruture, before switching the ECU into the `re-programming` mode. Furthermore, we assume that V2I infrastructure is available through out the firmware update process.

ECU Reprogramming Mode: If the type is the expected one, the diagnostic tool forces the ECU to switch from an application mode into a reprogramming mode by requesting a seed (`Na`). This seed is required to calculate an ECU specific key value to unlock the ECU for reprogramming. ECU verifies desired security properties. If it is true, ECU sends a HSM command `SecM_Generate(seed)`[7] to generate a seed. It then encrypt the seed $\epsilon\,(Na)_{M_k}$ for confidentiality enforcement, calculate a MAC on $\overrightarrow{m} = (\epsilon\,(Na)_{M_k} + Ts)$ and transmit it to the DT. At the same time, the ECU sends a HSM command `SecM_ComputeKey(Na, SecM_key)` to compute the key on the HSM with the aid of the generated `Na`. As output, the function delivers a $SecM_{key}$ key handle, we write $SecM_{key} = S_{mk}$, that is used to `unlock` the ECU.

Algorithm 2 ECU Reprogramming Mode

```
Require: DT and ECU have established a fresh and authentic connection based on a $M_k$. Vehicle is stopped and have
    access to V2I infrastructure
Ensure: Authentic and confidential exchange of ECU unlock key.

    1. DT  $\rightarrow$ ECU: $\left\{(request\_seed, Ts)\,,\ \{\widehat{m}\}_{M_k}\right\}$

    2. ECU: $\rightleftarrows$ HSM:: SecM_Generate($seed$)
       DT  $\leftarrow$ ECU: $\left\{\left(\epsilon\,(Na)_{M_k}\,, Ts\right),\ \{\widehat{m}\}_{M_k}\right\}$
       ECU: $\rightleftarrows$ HSM:: $S_{mk}$:= SecM_ComputeKey(seed, $SecM\_key$)

    3. DT: $\rightleftarrows$ HSM:: $S_{mk}$:= SecM_ComputeKey(seed, $SecM\_key$)
       DAT: $Exported\_S_{mk}$ := $key\_export$( $T_k$-handle = <$M_k$-handle>, kh=< $S_{mk}$-handle>)
       DT  $\rightarrow$ ECU: $\left\{(Exported\_S_{mk}, Ts)\,,\ \{\widehat{m}\}_{M_k}\right\}$

    4. ECU: $\rightleftarrows$ HSM:: $SecM\_CompareKey\,(key, seed)$
       DT  $\leftarrow$ ECU: $\left\{(ACK, Ts)\,,\ \{\widehat{m}\}_{M_k}\right\}$
```

In the diagnostic tool, verifies $\{\widehat{m}\}_{M_k}$, decrypt the received seed $(\epsilon^{-1}\,(Na)_{M_k})$ and compute the S_{mk} with the aid of the received seed (`Na`). Once the S_{mk} key value is computed, it is exported, using M_k as a transport key, and transmitted to the target ECU. The ECU verifies the $\{\widehat{m}\}_{M_k}$ and compare the received S_{mk} key with the self-generated S_{mk}. If the two values are identical, the ECU switched into `unlock` state (from

[7]SHE command defined in the HSM

application mode to the reprogramming mode) and send an `ACK` message to the DT (see Algorithm 2). This message is sent after the ECU is switched[8] into the `unlock` state to make sure the switch has been performed. If the comparison failed, the flashloader[9] hold the ECU in `locked` state. Only in the unlock state ECU `reprogramming` is possible.

Firmware Encryption Key Exchange: In this phase we are considering two possible scenarios of exchanging firmware encryption keys: `i). on-line solution` and `ii). off-line solution`. In the on-line solution: service station has access to an online infrastructure of the manufacturer, it can request the firmware and as well as firmware encryption key – (SSK). The SSK is a stakeholder symmetric key pair[10], externally created, with `use_flag = decrypt`, key for stakeholder individual usage e.g., software update. Whereas, in the case of off-line, firmware is encrypted with pre-installed SSK.

Considering current trends and advancements in the automotive industry, on-line solutions provide more reliability, flexibility and will eventually increase the security of the on-board network. Sharing firmware encryption key only with specific ECUs makes an on-line solution more robust and generic from off-line approaches, where each vehicle share unique symmetric keys, that is pre-installed in the vehicles. Moreover, considering security levels [55], we argue to specify a key validity period (short term or long term keys) of the SSK, for an individual ECU. Both have its advantages and disadvantages. We suggest to use short term keys for firmware encryption. Short terms keys will expire after a short amount of time and thus, we see no need for instant revocation, if keys are compromised. This has the advantage that OEMs do not have to go through another key migration (installing new keys) process, if keys are compromised. Thus, in the following section we present the on-line solution.

Algorithm 3 Firmware Encryption Key Exchange

```
Require: On-line access to OEM server and PKI infrastructure
Ensure: Authentic and confidential firmware encryption key exchange between OEM and ECU

  1. DT  → OEM: $\{(request\_firmware\_encryption\_key, Ts), \{\widehat{\sigma}\}_{Sk\_dt}\}$

  2. OEM: $Exported\_SSK$ := $key\_export$($T_k$-handle = ⋖ $P_k$_ecu-handle| Psk-handle⋗, kh=⋖ SSK-handle⋗)
     DT  ← OEM: $\{(Exported\_SSK, Ts), \{\widehat{\sigma}\}_{Sk\_oem}\}$

  3. DT  → ECU: $\left\{(Exported\_SSK, Ts), \{\widehat{m}\}_{M_k}\right\}$

  4. ECU: ⇄ HSM:: SSK-handle := $key\_import$ ($T_k$-handle = ⋖ $S_k$_ecu-handle| Psk-handle ⋗, kh= ⋖ SSK-handle
     ⋗
     DT  ← ECU: $\left\{(ACK, Ts), \{\widehat{m}\}_{M_k}\right\}$
```

After successful reprogramming access at the ECU level, the DT sends a request (`request_firmware_encryption_key`), to the OEM server to get the firmware encryption key (see Algorithm 3). This request includes information about the ECU (i.e., ECU

[8]The information whether a re-programming request has been received or not shall be stored in non-volatile memory, e.g. EEPROM. Since switching from the application to the reprogramming mode shall be done via a hardware reset, all contents of volatile memory will be lost.

[9]Flashloader stands for the complete software architecture needed for in-system reprogramming of an ECU [46]

[10]We use the term Key Pair for those symmetric keys that feature complementary functions (sign/verify) or (encrypt/decrypt)

type, ECU identification number, firmware version, etc.). In the OEM side, verifies the authenticity and integrity of the received message. If this is true, OEM server retrieve the Pk_ecu from the Public Key Infrastructure (PKI), (possibly) maintained by an individual OEM, and export SSK using P_k_ecu as a T_k. This is only feasible if ECU is equipped with full HSM. In the case of the medium or light HSM-ECUs, the pre-shared key Psk will be used as a T_k. The OEM server export the SSK and send a signed message to the DT. As the SSK key blob is encrypted with the ECU key. It is not possible for the DT to retrieve the firmware encryption key. Next, the DT transmits the received firmware encryption key to the ECU. ECU import the SSK in the HSM using `key_import` function. The `key_import` function provide the assurance to the ECU that they key is generated by the OEM, by verify the authentication code send along with the encrypted key, and can only be decrypted by the specific ECU key. After importing SSK in the ECU-HSM, ECU sends an `acknowledgment` message about successful import of the SSK key.

Firmware Download: Once the SSK is successfully imported in the ECU-HSM, the DT sends the received signed and encrypted firmware ($\mathbb{F}_{rm}$) along with its ECR[11] reference: $\left\| sig\left(\mathbb{F}_{rm}, \mathbb{E}_{cr}, Ts\right) \rightarrow \widehat{\sigma_{\mathbb{F}_{rm}}} \overset{enc}{\rightarrow} \epsilon\left(\widehat{\sigma_{\mathbb{F}_{rm}}}\right)_{ssk} \right\|$, from manufacturer, to the Random-Access Memory (RAM) of the ECU. Following HSM `use_flag` approach, where multiple key-properties may be set, only the OEM server can sign and encrypt the firmware, whereas the receiving ECU can decrypt and verify the received firmware, using the same key material. The encrypted firmware is downloaded block by block (logical block). Each of those blocks is divided into segments, which are a set of bytes containing a start address and a length. The start address and the length of each segment is sent to the HSM during the segment initialization. For one block, a download request is sent from the DT to the HSM. HSM initializes the decryption service and sent an answer to the DT. The download then starts segment by segment. After sending last firmware segment, DT sends a `transfer_exit` message to the ECU (see Algorithm 4).

Algorithm 4 Firmware Download

```
Require: Signed and encrypted firmware from OEM
Ensure: Authentic, fresh and Confidential firmware downloaded in the ECU

  1. DT → ECU: $\left\{\left(\epsilon\left(\widehat{\sigma_{\mathbb{F}_{rm}}}\right)_{SSK}, Ts\right), \{\widehat{m}\}_{M_k}\right\}$

  2. ECU ⇄ HSM: SecM_InitDecryption($\epsilon\left(\widehat{\sigma_{\mathbb{F}_{rm}}}\right)_{ssk}$)
     HSM:SecM_Decryption($\varepsilon^{-1}\left(\widehat{\sigma_{\mathbb{F}_{rm}}}\right)_{ssk}$)
     DT → ECU: $\left\{\left(request\_transfer\_exit, Ts\right), \{\widehat{m}\}_{M_k}\right\}$

  3. ECU ⇄ HSM:SecM_DeinitDecryption()
     DT ← ECU: $\left\{\left(ACK, Ts\right), \{\widehat{m}\}_{M_k}\right\}$
```

Firmware Installation and Verification: For an installation of the firmware, we consider standard firmware installation procedure defined in [46], where each logical block is erased and reprogrammed. However, before the flash driver can be used to re-program an ECU, its compatibility with the underlying hardware, the calling software environment and with prior versions of the firmware has to be checked. This compatibility check is

[11]ECR = ECU Configuration Register, similar to the PCRs inside a TPM.

performed by means of a version information stored in the HSM monotonic counters. The HSM `Read_Counter` function is used to read out the value of a counter. The counter is referenced by a $counter\_identifier$ previously increased, after every authentic and successful installation of the firmware. These monotonic counters are defined to perform such a checking of its current version against the new firmwares version in order to prevent the downgrading attack to older firmwares.

For the verification, we defined two step verification process: In the first step, before `re-programming`, ECU verifies the signature of the firmware data. This is verified by using the pre-installed Manufacturer Verification Key MVK, proves that the software was indeed released from the OEM. In the second step: we construct a most minimal trusted computing base (TCB) during the installation phase. We compute an ECR trusted chain at each step of the firmware installation. The ECR reference is needed to ascertain the integrity/authenticity of the firmware data. An `Extend_ECR` function is defined to build the ECR trusted chain. This function is used for updating the ECR with a new (hash) value. The new value is provided as input and chained together with the existing value stored in the ECR, using a hash update function. As output, the function delivers the updated ECR value. After a successful installation of the new firmware data, software consistence check is performed. The check for software dependencies shall be done by means of a callback routine provided by the ECU supplier. This check is done after reprogramming and before setting the new ECR reference. Next, the `Compare_ECR` function is called. This comparison can only be performed after all writing procedures for the logical block have been finished. This function allows the direct comparison of the current ECR with a reference ECR value received with the firmware. It is also possible that the ECR reference may be contained inside the firmware itself. In this case the flashloader shall call a routine provided by the ECU supplier to obtain the ECR reference. if the check succeed, HSM `Preset_ECR` function is called. This function is used to manage references to ECR values by ECR indices in the context of secure boot. After successfully setting the ECR value, HSM (`Increment_Counter`) function is called to increment the monotonic counter with the new value. In the last step, the actual hardware reset is executed, the flashloader delete (i.e. overwrite) the routines for erasing and/or programming the flash memory from the ECU's RAM [46]. Making sure those routines are not present on the ECU in application mode. After the reset the application is started.

Discussion – Error Handling: Each function of the HSM returns a status after its successful or unsuccessful execution. Some functions may deliver further function specific error codes. The value of the status shows the positive execution of the function or the reason for the failure. In the case of failure, the flash process must stop with an error code and the ECU enters the `locked` state.

Summary: We have presented a firmware update protocol for a new security architecture that is deployed within the vehicle. We showed how a root of trust in hardware can sensibly be combined with software modules. These modules and primitives have been applied to show how firmware updates can be done securely and over-the-air, while respecting existing standards and infrastructure. In contrast to existing approaches, the protocols presented in this paper show the complete process, which involves service provider, vehicle infrastructure as well as manufacturer and workshop. By using secure in-vehicle communication and a trusted platform model, we show how to establish a secure end-to-end link between manufacturer, workshop and vehicle. Despite the fact that a trusted platform model poses certain constraints, such as cryptographic keys bound to a boot configuration, we showed how the protocols presented take these constraints into account by updating platform reference registers used during the boot phase of an ECU.

3.7 Updates and Keys in HSM

If keys that are stored and handled inside the HSM, these may be bound to a certain configuration of the platform (cf. auth-flags ECU Configuration Registers (ECRs)). This configuration inherently depends on the installed firmware. Through an update procedure, these keys may loose their validity, depending on the binding. We present a number of solutions to the problem:

Key Replacement All keys that are bound to a specific platform are exchanged by the corresponding certificate agency. This requires an online infrastructure as well as heavy off-board protocols and results in non-deterministic update times.

Key Updates The fact that an update procedure may only be triggered in an authorized and authenticated way guarantees that a successive version is similar to the original system, for which a key was given out. By heavily re-designing the key structure, key-updates based on re-signing the original key material with adjusted authorization flags may be installed. This, however, imposes new security risks as the updater will indirectly have the same trust level as *any* other certification agency. Thus, the approach is rather theoretic.

Bind key to early boot phase A pragmatic way to achieve the usability of configuration-bound keys after an update is to bind them to an early configuration stage, e.g. after the bootloader has finished the authentication process of firmware. A potential disadvantage is that the bootloader itself must not be changed during updates, which is, however, rarely the fact. If it is changed, the affected keys have to be changed.

Additional ECR flag An additional binary ECR flag, that reflects simply the state "securely booted", can be introduced. The usage of this flag strongly depends on the security of the update process. Also, keys would not be bound to a specific device, which obsoletes the idea of the ECR authentication flag.

Summarizing, we assume that the bootloader is not changed in typical updates and thus will not affect authorization flags of HSM keys.

3.7.1 Replacement of Hardware and Sensors

We assume: A sensor s has only a light HSM installed, i.e., it only provides symmetric encryption. A secret is already installed in the HSM. This is called IDK_s, referred to as k_s in the following. The sensor is to be installed at an official workshop that has either an online connectivity to the manufacturer or is part of the manufacturer's certification program (i.e., it has a valid certificate).

The problem: The key k needs to be transported into the KeyMaster of the respective domain of the sensor in a secure, i.e., authentic and confidential, manner, however, k can not be exported outside the HSM (i.e., it must not be exported without transport encryption, but there exists no shared key for transport encryption).

The solution: As workshops may be equipped differently (think of dealerships in cities and rural areas), we saw the need to specify an on-line and off-line approach. They rely on different trust anchors: The on-line method provides the trust from a manufacturer back-end, whereas the off-line solution incorporates a tamper-proof piece of hardware, approved by the manufacturer.

On-line: If the garage has access to an online infrastructure of the manufacturer, it can request the sensor's key k. This would be delivered inside an authenticated and certified (i.e., signed with the MVK) key blob (Binary Large Object), that is encrypted with the corresponding public IDK of the KeyMaster, to which k is to be installed. This specific IDK public key would need to be transmitted to the manufacturer in the request. The received key-blob can then be imported by the corresponding KeyMaster. An abstracted view is given in Figure 17, where the new sensor ECU_1 is installed.

Off-line: The key k must be provided to the garage g along with sensor s. The garage uses certified diagnosis equipment. The diagnosis equipment contains an HSM that holds the garage's secret key SK_g. The key-blob provided with the spare part contains the signed (with manufacturer key) cryptographic key k_s encrypted with the public key of the garage. The diagnosis equipment requests the KeyMaster's public key and uses it to re-encrypt k_s and its certificate. The KeyMaster imports the key, as it is signed by the manufacturer.

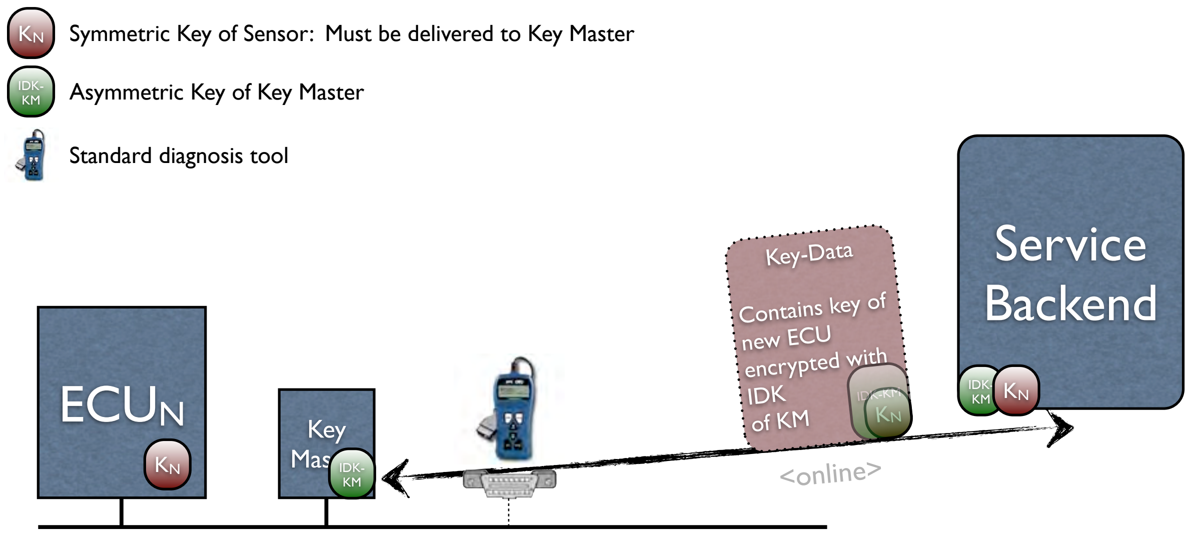

KN
Symmetric Key of Sensor: Must be delivered to Key Master
IDK-KM
Asymmetric Key of Key Master
Standard diagnosis tool
ECUN
KN
Key Master
IDK-KM
Key-Data
Contains key of new ECU encrypted with IDK of KM
IDK-KM
KN
<online>
Service Backend
IDK-KM
KN

Figure 17 Replacement of ECU_1 and deployment of corresponding PSK at KeyMaster using an online back-end infrastructure

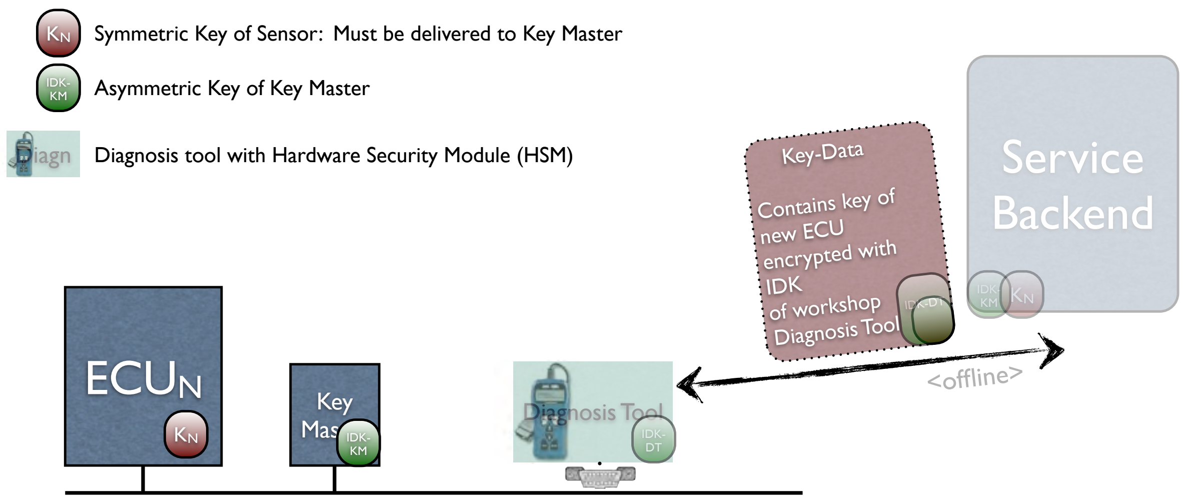

Symmetric Key of Sensor: Must be delivered to Key Master
Asymmetric Key of Key Master
Diagnosis tool with Hardware Security Module (HSM)
Key-Data
Contains key of new ECU encrypted with IDK of workshop Diagnosis Tool
Service Backend
ECUN
KN
Key Master
IDK-KM
Diagnosis Tool
IDK-DT
<offline>

Figure 18 Replacement of ECU_1 and deployment of corresponding PSK at Key-Master using a cryptographic envelope for the diagnosis tool equipped with an HSM. An online connection to the backend is not required.

3.7.2 Secure Device Integration Protocol

In order to securely attach and integrate a mobile device to the vehicle, we propose the following protocol (see Figure 19) . The protocol specifically initializes an authentic and confidential communication channel between the mobile device and the vehicle. Therefore, a mutual authentication is performed between the vehicle and the mobile device. According to the supported authentication protocol, a dedicated EAM plug-in is selected. Before the authentication is processed, it is checked whether the entities have the respective authorizations. After successful authentication, a session key for confidential communication is established.

In addition to the setup of an authentic and confidential communication channel between vehicle and mobile device, the communication can be filtered according to the requirements of the application. This is besides authorization, part of the service proxy functionality inside the vehicle. The service proxy can additionally filter messages (firewall), either stateful or non-stateful, detect malicious behavior by interfacing with the SWD and eventually initiate certain countermeasures (intrusion response) to the system. The configuration of the service proxy highly depends on the respective application, e.g. CE device integration, access of certain functionality or data, etc. Therefore, corresponding authorization, filter and intrusion detection and response policies need to be specified.

3.8 Transport protocol with security features

3.8.1 General Approach

In order to have a transport protocol compliant with CAN and FlexRay, we propose the use of the Common Transport Protocol (CTP) already known from the EASIS project [43].

Regarding the security features, we have to keep in mind that we only take care for

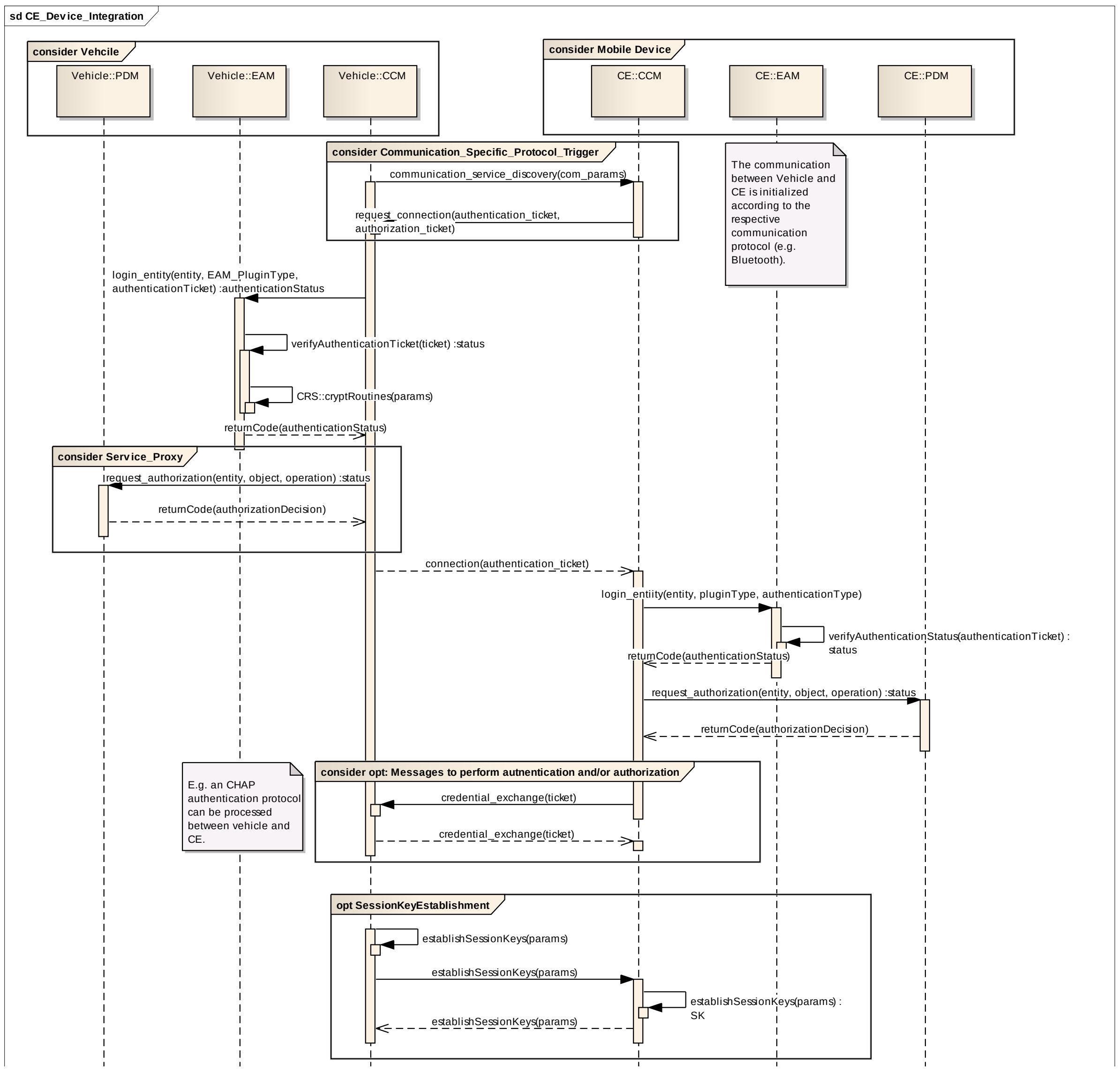

sd CE_Device_Integration
consider Vehcile
Vehicle::PDM
Vehicle::EAM
Vehicle::CCM
consider Mobile Device
CE::CCM
CE::EAM
CE::PDM
consider Communication_Specific_Protocol_Trigger
communication_service_discovery(com_params)
request_connection(authentication_ticket, authorization_ticket)
The communication between Vehicle and CE is initialized according to the respective communication protocol (e.g. Bluetooth).
login_entity(entity, EAM_PluginType, authenticationTicket) :authenticationStatus
verifyAuthenticationTicket(ticket) :status
CRS::cryptRoutines(params)
returnCode(authenticationStatus)
consider Service_Proxy
request_authorization(entity, object, operation) :status
returnCode(authorizationDecision)
connection(authentication_ticket)
login_entiity(entity, pluginType, authenticationType)
verifyAuthenticationStatus(authenticationTicket) : status
returnCode(authenticationStatus)
request_authorization(entity, object, operation) :status
returnCode(authorizationDecision)
E.g. an CHAP authentication protocol can be processed between vehicle and CE.
consider opt: Messages to perform autnentication and/or authorization
credential_exchange(ticket)
credential_exchange(ticket)
opt SessionKeyEstablishment
establishSessionKeys(params)
establishSessionKeys(params)
establishSessionKeys(params) : SK
establishSessionKeys(params)

Figure 19 Message sequence diagram for the secure attachment of CE devices

a secure on-board communication, but not for secure external communication. For on-board communication, the security shall not be implemented on transport layer. Instead, it makes more sense to implement the security on higher layers. This gives more flexibility since the requirements for data transfer between individual components inside the vehicle are quite different. In most cases, only authenticity and integrity are required, but no confidentiality. This means that the encryption features of a common security protocol would be unused most of the time. By this way, the design approach of the transport protocol addresses the key objectives of efficiency, flexibility and scalability.

For our approach, this means that the application of security features is not a direct component of the transport protocol. Instead, we will foresee the possibility to apply the requested security to the payload data which then may contain a MAC and time information and also contain encrypted data. Individual security requirement shall be determined by the individual application and then be enforced by the CCM when preparing the payload data for transmission.

In the next subsections, we first describe the design of the transport protocol independently of security features. Finally, the approach for applying security to the payload data is described separately in Section 3.8.5.

3.8.2 Use of CTP Features for Message Broadcasting

Although the aforementioned use cases only foresee the transfer of certain messages from one sender to one receiver, today's on-board communication does not normally use such one-to-one ad-hoc communication. Instead, a signal-based communication is used: Each message is sent over the on-board network and equipped with a certain ID, and each ECU determines the relevance of a message according to this ID (acceptance filtering). This approach avoids for example the complexity of having to send the same message to each ECU, for which the message could be relevant and uses the medium's physical broadcast functionality. This is essentially providing data in a periodic manner to a certain group of users (multicast group).

To be in line with this automotive approach, the transport protocol needs to be designed for group communication between different ECUs. This is foreseen in the CTP description in [43]. Furthermore, as described in Appendix A, CTP provides the flexibility to use flow control or final acknowledgments, which can be necessary, depending on the application: e.g. for applications with inter-domain communication involving different transmission capacities or for the transmission of multi frame messages. The decision is made by using different frames types. Nevertheless, flow control and acknowledgments are only applicable to unicast messaging but not to broadcast messaging. Therefore, we will not use these explicit CTP features but also implement flow control and acknowledgment optionally on application level. By this way, the individual application decides whether the features are necessary or not.

Our proposed solution keeps the general structure of the CTP header but just uses one of the fields with a different meaning. It shall still be usable independently of the data link layer protocol, i.e. in the same way over CAN and FlexRay.

According to [43], the CTP header contains a 15-bit source address and a 15-bit target address indicating the addresses of sender and receiver. For CTP over CAN, most parts of these address fields are coded within the 29-bit CAN ID, and some additional bits not fitting into the CAN ID are coded within the payload part. For CTP over FlexRay, the whole CTP header is a part of the payload data.

For our transport protocol, these address fields shall still be present with the following meanings: The source address field indicates the address of the sending ECU as also foreseen by the initial CTP specification. The 15-bit target address field is now used as a message identifier containing the destination address, by which relevance for individual ECUs (participating in a certain recipient group) is given.

For CTP over CAN, [43] first discusses three different variants how to extend the 11 bit addresses from (original) CAN IDs into 15-bit addresses by using one byte of the payload data. Then the final decision is to use the variant with the source address completely contained in the CAN ID. For our purpose, it is also reasonable to use this variant since the determination of the intended target address is basically done on the base of the content of the message.

Figure 20 shows the bit structure of a CAN frame for our transport protocol. In this notation, the data part consists of `EVITA_data` (see notations above or 3.8.6 for details) as well as the Network Layer Protocol Information (N-PCI) (see later section on fragmentation).

29-bit CAN ID			Payload	
28 ... 22	21 ... 7	6 ... 0	63 ... 56	55 ... 0
reserved	Source Address	Message Identifier		Data

Figure 20 Bit structure of CAN frame for transport protocol

Although the specification of CAN does not exclude the possibility to use the same CAN ID for different senders, our protocol design using the idea of the complete source address as part of the CAN ID by extending the address space, provides unique address scheme, i.e., no two different ECUs are sending messages under the same CAN ID. In case of FlexRay, source address and message identifier are parts of the payload data in the same way as defined in [43].

3.8.3 Concept for Message Transfer

We want to assume the situation after execution of the key distribution according to 3.2.3, i.e. session keys for group communication (with separate keys for encryption and MAC generation) have been agreed and been installed by `CCM_Open_Channel`. We now have the provisions for secure group communication, and we want to use these features for message broadcasting within the group of ECUs. The real message transfer will be performed in the following way:

- The sending ECU (denoted ECU_1) sends the message to the whole on-board network by `CCM_Send_Message`. The intended target ECU (number k) is indicated by the message identifier field in the CTP header.
- Security features are applied according to the security policies set by `CCM_Open_Channel` and with the aid of keys that have previously been agreed or are exchanged ad-hoc.
- The intended target ECU_k receives and processes the message, all others ignore it.
- If response data are needed, ECU_k now takes the role of the sending ECU and performs the same actions with the response message with ECU_1 as the intended receiving ECU.

As far as required by the application, flow control and acknowledgment will be handled with the aid of optional response messages: If the receiver is supposed to give an acknowledgment (only for one-to-one communication with shared keys), it formats a corresponding response message whose content gives the necessary feedback. This message is secured with the MAC analogously. No special concept for acknowledgment messages needs to be specified.

The security features are applied and coded together with the data format by the function `CCM_Send_Message`. More details will be given in Section 3.8.5. For each message, the sender plays the role of the master ECU. The on-board network does not contain a general master gateway responsible for the distribution of all internal messages, but it can contains gateway responsible for a particular domain in the car. The transfer of the message from CTP over CAN, FlexRay and IP is described in Appendix A.

3.8.4 Fragmentation

Depending on the message length, the message may either be sent within one frame or needs to be fragmented into several frames. We can basically use the same CTP frame types already defined in [43]. However, some of them are obsolete since we do not use flow control and acknowledgment for group communication.

For our purpose, we just need the following frame types:

- one frame type `SF` for single frame messages (not requiring any acknowledgment)
- the frame types `FF-1B` – `FF-4B` for first frames with 1–4 byte length coding
- a frame type `CF` for consecutive frames.

Similar as in [43], the N-PCI consists of a 1-5 byte data element whereby the first byte indicates the frame type indication (FID) or the sequence number, SN, for CFs, and the following ones the length of the message (data length, DL). The collection of our frame types is summarized in Table 23

For CAN as well as for FlexRay, the N-PCI is a part of the payload data of the frame. The real application data (`EVITA_data`) then consist of the remaining 0 to 6 Bytes in case of CAN and up to 26 bytes in case of FlexRay – see Figures 21 and 22.

Data (max 8 bytes)		
1 byte	1...5 bytes	0...6 bytes
msg id extension	N-PCI	EVITA_data

Figure 21 EVITA application data embedded into CAN payload data

Data (max 35 bytes)				
2 bytes		2 bytes	1...5 bytes	0...26 bytes
op	msg id	src address	N-PCI	EVITA_data

Figure 22 EVITA application data embedded into FlexRay payload data (op=indication bit for odd number of bytes, see [43] and Appendix A.4)

3.8.5 Use of Security Features

Communication is secured according to certain security properties that an application requests. Security features are available via the CRS as defined in [64]. However, they are neither integrated into the transport protocol nor explicitly implemented by a security protocol on application layer (OSI layer 7). Instead, the security properties are requested via the CCM of the security framework. The security properties modify the payload of data packets in the way that:

Frame Type	1st byte	2nd byte	3rd byte	4th byte	5th byte	Description
SF	FID	DL	–	–	–	Single Frame
FF-1B	FID	DL	–	–	–	First Frame 1-byte length encoding
FF-2B	FID	DL		–	–	First Frame 2-byte length encoding
FF-3B	FID	DL			–	First Frame 3-byte length encoding
FF-4B	FID	DL				First Frame 4-byte length encoding
CF	SN	–	–	–	–	Consecutive Frame

Figure 23 Structure of N-PCI – reduced set of CTP frame types

- MACs are added if *integrity* is needed,
- Payload Encryption is added if *confidentiality* is needed,
- Time information (tick- or timestamps) is added if *freshness* is needed. It is bound to the payload and requires integrity as additional security property.

Depending on the application, the CCM functions automatically call the functions `Gen_MAC` and `Encrypt` for message encryption and MAC calculation. The receiving ECU detects the security features of the payload data, and then calls the functions `Ver_MAC` and `Decrypt` during message processing if necessary.

As stated below, the security features finally become a part of the payload data. Nevertheless, we want to exactly specify a unique format for the payload data of messages uniquely indicating the security features applied to a message. We now need to make a tradeoff how much effort we invest into the coding of the payload data:

- On the one hand, the presence of a MAC, a timestamp or a cryptogram as well as the cryptographic algorithms and the size of the MAC fraction must be visible,
- On the other hand, the amount of data to be transferred must not get too large by the use of complicated data structures, e.g. TLV (Tag, Length, Value) objects.

The conditions which security functions apply are already set by the function `CCM_Open_Channel` during session establishment. But on receiver's side, we need to provide appropriate control mechanisms whether these security features have really been applied for message transfer. The cryptographic material that is necessary for these operations is agreed by keying protocols as defined earlier.

We want to introduce a message format that clearly indicates the applied security features including the chosen cryptographic algorithms and the length of the MAC fraction. To avoid additional data fields, we offer a limited choice for each parameter that may be coded as short bit flags within one control byte. Regarding cryptographic algorithms, we relate to [64], i.e. we have AES-128 for symmetric encryption and WHIRLPOOL (HMAC), SHA-256 (HMAC) and AES-128 (CMAC) as potential MAC algorithms. For the length of the MAC fraction, we offer the choice between four different absolute values, starting from the minimum length of 32 bits.

For our security concept, we look at the message as a whole which may need to be transmitted via fragmentation by several CTP frames. The application on sender's side secures the whole message before transmission, and the application on receiver's side first collects the data from all received frames and then performs the final operations of decryption and MAC verification. This in particular means that we will only have one control byte transmitted by the first frame and at most one MAC and one time stamp transmitted by the last consecutive frames.

3.8.6 Structure of EVITA_data

Security shall only be applied to the real application data (`EVITA_data`), but not to any address data, the message id and the N-PCI (see Figures 21 and 22). The control byte itself will be part of `EVITA_data`, but also not covered by encryption or MAC protection.

If a message is encrypted, the encryption shall also cover the MAC as well as the timestamp, and the timestamp itself is also covered by the MAC. The structure of the `EVITA_data` (typically to be transferred over several frames) is shown in Figure 24. The length field of the first or single CTP frame contains the complete length of the `EVITA_data` according to Figure 24.

	plain text	plaintext or cryptogram		
Object	Ctrl Byte	Message Data (m)	time stamp	MAC
Length	1 Byte	variable	fixed	fixed
	mandatory		optional	

Figure 24 Format of complete message payload (EVITA_data) containing security features (may be fragmented over several frames)

The control byte indicating the applied security features shall be composed as indicated in Figure 25. The leading bit of the control byte as well as two codes for the encryption algorithm are currently not used (Reserved for Future Use (RFU)), i.e. this specification is open to be enlarged for coding further message features. It is a mandatory byte to be found in plain text at the beginning. After that, the real message data (with variable length) are following. Their length is determined by the length field of the CTP header together with the control byte. If a MAC – with or without a time stamp – is applied, the data structure contains these two data fields as additional components. Their length is uniquely determined by the control byte (since only dependent on the choice of algorithms and MAC fraction length). If encryption is used, the whole part of `EVITA_data` except the beginning control byte is present in encrypted form.

The steps of processing the payload data are then integrated into the execution of the function `CCM_Send_Message` as shown in Figure 26.

Please note: The address identifiers for sender and receiver shall be unique per application (i.e., service oriented adressing). If this can not be guaranteed, an additional application identifier needs to be embedded into the signed payload.

Please note as well: On the CAN bus the addresses are not directly part of the MAC. Their authenticity is validated indirectly though the key-handle, which is closely linked to the recipient-group. It is guaranteed by platform integrity and secure storage (see the section below) that this association can not be manipulated.

Bit No.	7	6	5	4	3	2	1	0	
	0	0	1	x	x	x	x	x	encryption with AES-128
	0	0	0	x	x	x	x	x	no encryption
	0	x	x	1	1	x	x	x	use of WHIRLPOOL HMAC
	0	x	x	1	0	x	x	x	use of SHA-256 HMAC
	0	x	x	0	1	x	x	x	use of AES-128 CMAC
	0	x	x	0	0	x	x	0	no MAC (Bit 2 and 1 ignored)
	0	x	x	x	x	0	0	x	32 Bit MAC fraction
	0	x	x	x	x	0	1	x	64 Bit MAC fraction
	0	x	x	x	x	1	0	x	96 Bit MAC fraction
	0	x	x	x	x	1	1	x	128 Bit MAC fraction
	0	x	x	x	x	x	x	1	use of timestamp (only if bit 4 or 3 are set)
	0	x	x	x	x	x	x	0	no timestamp
	all other values								RFU

Figure 25 Structure of control byte indicating security features

3.9 Secure Storage Protocol

3.9.1 General Scope and Assumptions

During operation of the on-board network, data is transported between ECUs and needs to be securely stored inside the destination ECU. In the following, we specify a protocol for such secure data storage. Distribution and storage of keys is out of scope here since this has been covered separately within Section 3.2. We only look at the process of storing data inside the destination ECU and assume that the data has been securely transported using the transport protocol specified in Section 3.8.

Secure storage is achieved by creating a symmetric cryptographic key, restricted to ECRs that identify a securely booted platform. Different methods of access control (see Section 3.4) are applied. A number of access control measures can be taken.

3.9.2 Key Access Control

Depending on the level of storage (user data, application data or system data), different measures are taken:

- Use-flags of the key are bound to a securely booted platform via ECR (always),
- Use-flags of the key are bound to a specific execution context and bootstrap phase via ECR (system),
- Use-flags of the key structure are bound to an application-supplied credential (application-data and user-data),
- Secondary key material (explained below) is protected by standard operating system measures such as file permissions.

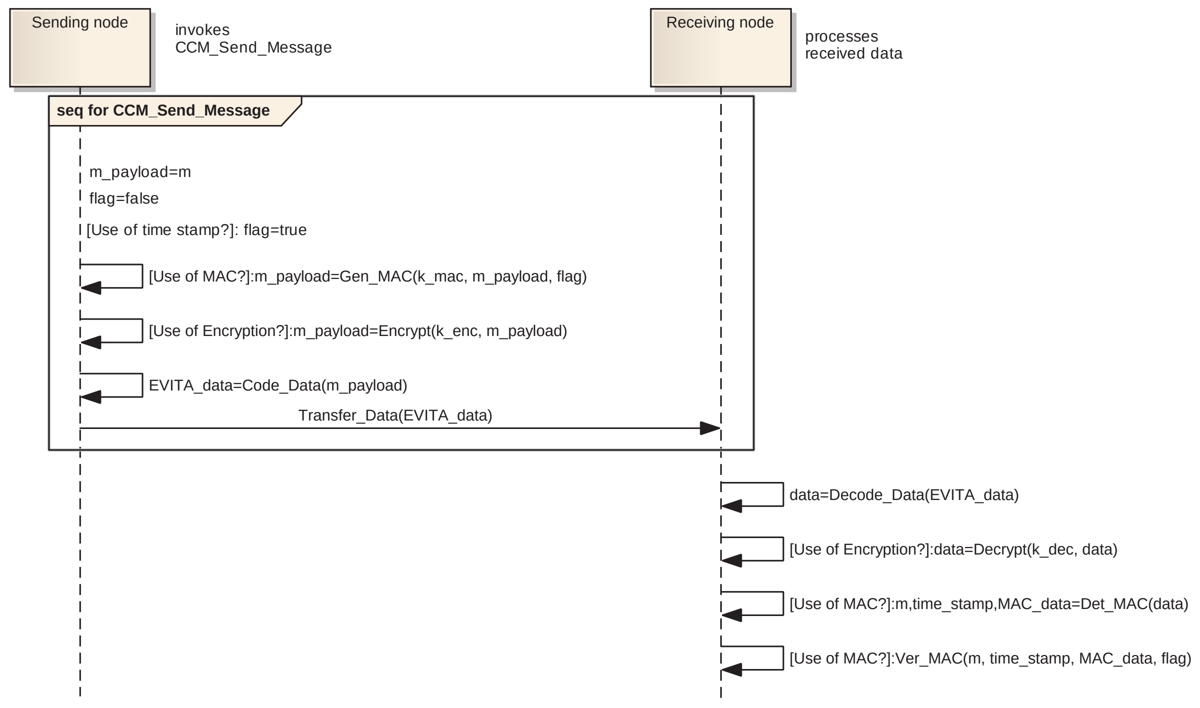

Sending node
invokes CCM_Send_Message
Receiving node
processes received data
seq for CCM_Send_Message
m_payload=m
flag=false
[Use of time stamp?]: flag=true
[Use of MAC?]:m_payload=Gen_MAC(k_mac, m_payload, flag)
[Use of Encryption?]:m_payload=Encrypt(k_enc, m_payload)
EVITA_data=Code_Data(m_payload)
Transfer_Data(EVITA_data)
data=Decode_Data(EVITA_data)
[Use of Encryption?]:data=Decrypt(k_dec, data)
[Use of MAC?]:m,time_stamp,MAC_data=Det_MAC(data)
[Use of MAC?]:Ver_MAC(m, time_stamp, MAC_data, flag)

Figure 26 Steps for incorporation of security by CCM_Send_Message and message processing by receiver

3.9.3 Native Encryption

This method uses the HSM directly. While this may be more efficient, it may limit the overall throughput for other applications utilizing the HSM. A symmetric key is generated with corresponding use flags (see above). It is then utilized using the CRS functions `CRS_encrypt_init()`, `CRS_update()`, and `CRS_finish()`. Output of the update function is sequentially written into a file on the local filesystem.

3.9.4 Secondary Encryption

A way to limit the use of resources of the HSM is to use a secondary key that is stored in non-volatile memory. This secondary key is stored using *Native Encryption*, i.e., only the secondary key is directly encrypted and decrypted by the HSM. The use of this key must be bound to a certain execution context so that platform integrity, especially concerning the algorithms used in software, is assured. In Figure 27 a schematic of this use is shown.

The method of secondary encryption is well suited for the realization of a hybrid encryption scheme making use of asymmetric cryptography provided by the HSM. In this case, the task of the HSM is limited to the encryption and decryption of the symmetric data encryption key.

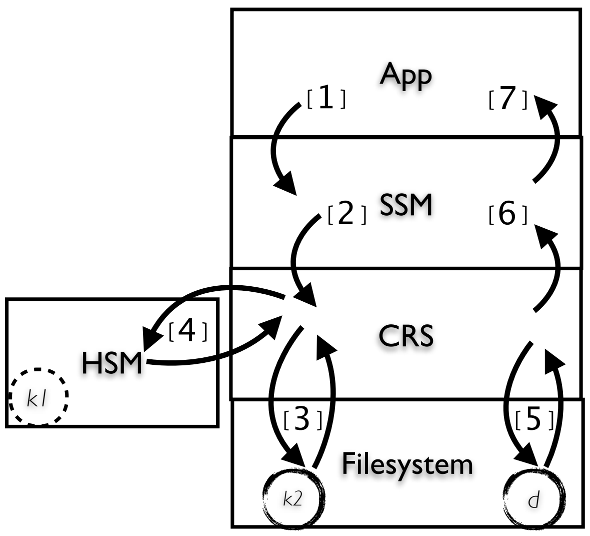


Figure 27 Secure storage in software: The actual key k_2 that encrypts data d is secured by a key guarded by the HSM (k_1). Data stored in the filesystem (k_2 and d) are protected by encryption, while k_1 is stored by the HSM internally. The schematic shows how the secure storage module requests data (step 1), which triggers the cryptographic module to retrieve and decrypt the corresponding secondary key (steps 2–4) and finally decrypts data (5) and delivers it to the requester (steps 6–7).

4 Simulation and Evaluation

The formal verification of the EVITA approach – including the hardware software infrastructure and security protocols – is the objective of ongoing work. This section presents first models and puts the emphasis on the evaluation of security protocols, using mostly simulation techniques, but also formal verification. This evaluation does not target an exhaustive verification of requirements, but rather focusses on a few metrics like performance and reachability of given states. The proposed models are similar to the ones used in [16].

From a modeling point of view, protocols can be identified wherever two or more entities need to interchange information. Entities are thus delimited by a common virtual, logical, functional or/and physical border. In a simplified approach, a protocol can be separately analyzed from other related elements, like for instance software and hardware components or mechanisms, thus focusing on particular characteristics like the information interchange. A magnified view is used for the protocol evaluation. Such approach has been already presented in [16]. On the one hand, a magnified view of an embedded and distributed system increases the complexity of the model since more details are considered in that view. But on the other hand, less elements of the overall architecture are considered since only the ones involved in the view under studied are modeled. Nonetheless, once model correctness is achieved for magnified views, we expect to transpose in the general view results from magnified views, and also accordingly directly modify the overall architecture if necessary.

Protocol modeling takes as input both the description provided in previous sections and also the infrastructure for protocol execution described in [64]. In [64], a wide range of hardware and software entities have been specified. In our approach, protocols are meant make use of the support offered by Security Software Modules as well as the operation of HSMs: among others, protocol evaluation targets the following property: *Protocols efficiently use software and hardware modules, and use them according to the way they are defined (i.e., they respect their respective APIs).* Such evaluation can be made according to functional aspects, like for instance required performance, energy consumption and time constraints satisfaction, and also according to security aspects, like for instance the related possible attacks due to insecure channel communication and the security requirements to prevent such attacks. Therefore the framework to perform that evaluation should have the capabilities to represent such various elements.

In this section, we describe the modeling and simulation of protocols, using the magnified view already proposed in [16]. This simulation helps to evaluate the protocol execution based on certain modeling assumptions and with respect to certain evaluation criteria that are accordingly presented and explained. Thus the simulation results provide evidence about how the protocol execution impacts the system performance as well as the (un)satisfaction of certain security properties that have been specified co-jointly to the model. So far, Unified Modeling Language (UML) profiles have been used for modeling protocols: the TURTLE profile [3], and the DIPLODOCUS profile [35]. The ins and outs of the two profiles are discussed in this section, in the scope of simulation and formal verification.

4.1 Protocol Representation in our Approach

In this section we describe how protocols are represented in our approach. We also introduce a Target of Verification (ToV) which instantiates the previously referred magnified view. This representation will be particularly useful to understand the TURTLE model Section 4.2.

4.1.1 CAM-LDW Protocol Representation

In our approach, each protocol is represented through the following sets: a set of *Agents* or *Entities*, the *Initial Knowledge* of each agent, the *Assumptions* related to the protocol and the description of *Information Interchange*. To ease model comprehension we first introduce some notations in Table 3 that are then further used in the rest of this section.

Notation	**Description**
EV	External vehicle.
CU_ECU	Communications Unit ECU. Analog notations are used for Chassis and Safety controller (CSC) and Head Unit (HU) ECUs.
CU_CRS	Cryptographic Services Module (CRS) in Communications Unit (CU). Analog notations are used for CRS modules of other ECUs.
CA	Certification Authority. A valid Certification Authority CA is recognized by the Communications Unit ECU for secure external communications.
$X - PuK$	The public asymmetric key of the agent X.
$X - PrK$	The private asymmetric key of the agent X.
$SesK$	A symmetric Session Key that is owned by a group of ECUs.
CAM	Cooperative Awareness Message which includes Local Danger Warning (LDW) information.
$Pseudo$	Refers to an entity or agent's pseudonym that is used to ensure privacy.
1 || 2	Actions 1 and 2 are performed in parallel.

Table 3 Notation for modeling section.

The CAM-LDW protocol is a representative protocol of the EVITA specification since it is directly related with the information that may be interchanged between vehicles (V2V communication) to inform and prevent dangerous situations.

Regardless the CAM-LDW Protocol has interactions with entities from several infrastructure layers, we can abstract such interactions and represent the protocol in a plain form as follows:

Agents: External Vehicle (EV), Communications Unit ECU (CU_ECU), Head Unit ECU (HU_ECU), Chassis and Safety Controller ECU (CSC_ECU).

Initial Knowledge: CU_ECU possesses the public key of a Certification Authority (CA) ($CA - PuK$). All the internal ECUs belongs to the same group. In consequence they own the same Session Key for MAC verification ($SesK$). Additionally, CU_ECU has the complementary part of the $SesK$ for MAC generation. The functionalities attached to each ECU correspond with those already specified in [64].

Assumptions: The CU_ECU has the public key of a valid CA which is used to verify certificates from external agents. The CAM message does not need confidentiality, however the anonymity of the vehicles' identities should be preserved. Therefore pseudonyms are used for external Communications Unit identification. As a matter of fact, a valid Session Key has been correctly distributed among the members of the same group of ECUs which permits that CU_ECU generates and sends authenticated messages to CSC_ECU and HU_ECU (see Section 3.2.3). The SW-HW module interactions inside of each ECU are hidden. Even though this simplification is made, the corresponding security operations are simulated. As many of those interactions rely on CRS module we model this component separately.

The information interchange of this protocol is shown in the next list.

1. $EV \rightarrow CU\_ECU$: $CCM\_send(Sig(CAM, time\_stamp, Pseudo - PrK),$ $Cert(Pseudo - PuK, CA - PrK))$

2. $CU\_ECU \rightarrow CSC\_ECU$: $CCM\_open\_channelREQ(CSC - ECU,\ SecPropertySet)$

3. $CSC\_ECU \rightarrow CU\_ECU$: $channel\_id$

4. $CU\_ECU \rightarrow CSC\_ECU$: $CCM\_send(channel\_id, [LDW, time\_stamp, MAC(LDW,$ $time\_stamp)])$

5.||2. $CU\_ECU \rightarrow HU\_ECU$: $CCM\_open\_channelREQ\ (CU - ECU, SecPropertySet)$

6. $HU\_ECU \rightarrow CU\_ECU$: $channel\_id1$

7. $CU\_ECU \rightarrow HU\_ECU$: $CCM\_send(channel\_id1, [LDW, time\_stamp, MAC(LDW,$ $time\_stamp)])$

The protocol is triggered in the CCU ECU whenever a Cooperative Awareness Message (CAM) is received. This message should contain CAM information as well as the respective time stamp. The message is signed with the private key of the anonymous external vehicle ($Pseudo - PrK$). The public key of the external Communications Unit ($Pseudo - PuK$) is provided in a certificate that is signed with the private key of a previously accepted CA (CA). Afterwards CU_ECU performs several security operations (see Figure 28); signature and certificate authentications, freshness verification, enforcement of Access Control Policies to ensure CA privileges and plausibility check of the LDW information. If the security operations lead to a valid Local Danger Warning, then the LDW is MAC ciphered and time stamped with the pre-shared Session Key. Afterwards, respective communications channels are requested to CSC_ECU and CU_ECU. Finally, the authenticated LDW is sent using the respective channels. Both ECUs verify the authenticity and freshness of the message and perform operations according to the nature of the LDW. A complementary sequence diagram is presented in Figure 28.

The second instance included in our model is the access to Controller Area Network bus. Because of the nature of CAM-LDW protocol, distribution of LDW messages must satisfy temporal constraints. Consequently, the access to the CAN bus should be included in the model. The CAN protocol is implemented in the lowest layer of the infrastructure. However, in the scope of our evaluation, a detailed model of the CAN bus is not at stake

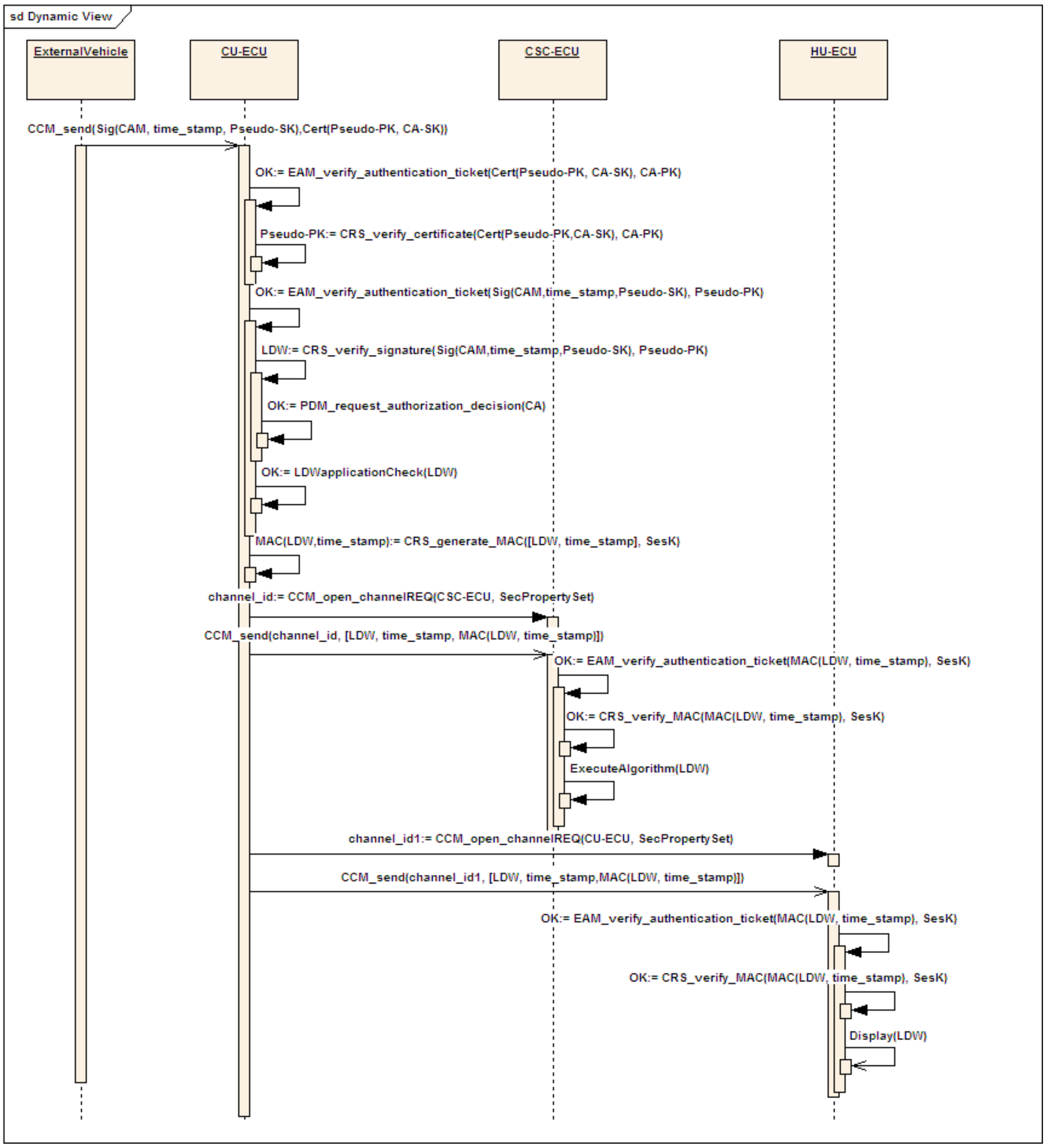

sd Dynamic View
ExternalVehicle
CU-ECU
CSC-ECU
HU-ECU
CCM_send(Sig(CAM, time_stamp, Pseudo-SK),Cert(Pseudo-PK, CA-SK))
OK:= EAM_verify_authentication_ticket(Cert(Pseudo-PK, CA-SK), CA-PK)
Pseudo-PK:= CRS_verify_certificate(Cert(Pseudo-PK,CA-SK), CA-PK)
OK:= EAM_verify_authentication_ticket(Sig(CAM,time_stamp,Pseudo-SK), Pseudo-PK)
LDW:= CRS_verify_signature(Sig(CAM,time_stamp,Pseudo-SK), Pseudo-PK)
OK:= PDM_request_authorization_decision(CA)
OK:= LDWapplicationCheck(LDW)
MAC(LDW,time_stamp):= CRS_generate_MAC([LDW, time_stamp], SesK)
channel_id:= CCM_open_channelREQ(CSC-ECU, SecPropertySet)
CCM_send(channel_id, [LDW, time_stamp, MAC(LDW, time_stamp)])
OK:= EAM_verify_authentication_ticket(MAC(LDW, time_stamp), SesK)
OK:= CRS_verify_MAC(MAC(LDW, time_stamp), SesK)
ExecuteAlgorithm(LDW)
channel_id1:= CCM_open_channelREQ(CU-ECU, SecPropertySet)
CCM_send(channel_id1, [LDW, time_stamp,MAC(LDW, time_stamp)])
OK:= EAM_verify_authentication_ticket(MAC(LDW, time_stamp), SesK)
OK:= CRS_verify_MAC(MAC(LDW, time_stamp), SesK)
Display(LDW)

Figure 28 CAM reception and LDW distribution

since our point of interest is only how the bus can be accessed, and the delays that access imply.

Our access model considers the real behavior of system. Indeed, Carrier Sense Multiple Access/Collision Detection (CSMA/CD) is a network access method in which multiple access can be performed and collisions can be detected; whenever a collision is detected the transmitter has to resend the previous data what is equivalent to no channel availability and no data transmission. This in turn provokes a retransmission attempt after certain back-off delay. Additionally we assume that the access to the CAN bus is regulated by a message priority policy which implies that higher message priorities preempt the transmission of lower ones. Consequently our model considers the following aspects:

- The access to the CAN bus may be delayed if the priority of the ECU message is lower than the priority of the message being transmitted on the bus.
- This event can be modeled by considering that the ECU sends a request for CAN availability which includes the message priority.
- An answer $busBusy == true$ indicates that a collision has been detected and a higher message priority in the bus. The transmission attempt is thus modeled.
- After a back-off delay a new attempt is performed by following the same approach.
- If the bus is not busy or the ECU message priority is higher than the priority of the message on the bus, then the message is transmitted.
- To be in compliance with the CSMA/CD scheme, the bus denial or assignation processes are performed in the model without the passage of time what reflects the nature of the CAN bus access.

The description of this access is presented just below.

Agents: CU_ECU, CSC_ECU, HU_ECU, CAN bus Controller (CAN_bus).

Initial Knowledge: Each active agent knows its assigned priority which is used to challenge the CAN bus for transmission; $priorityCU$, $priorityCSC$ and $priorityHU$. Each active agent knows the time delay that it has to wait before a new retransmission attempt ($waitCANavailability$).

Assumptions: The CAN bus is assigned with a permanent message traffic of 33%. We assume that the bus load impact the number of accepted messages in such way that some of them can not be transmitted due to higher priorities in the bus.

An instance of the information interchange for the CAN bus access is shown in the next list.

1. $CU\_ECU \rightarrow CAN\_bus$: $priorityCU$

2.||1. $CSC\_ECU \rightarrow CAN\_bus$: $priorityCSC$

3.||1. $HU\_ECU \rightarrow CAN\_bus$: $priorityHU$

4. $CAN\_bus \rightarrow CU\_ECU$: $busAssigned$

5.||4. $CAN\_bus \rightarrow CSC\_ECU$: $busBusy$

6.||4. $CAN\_bus \rightarrow HU\_ECU$: $busBusy$

7. $CU\_ECU \rightarrow CAN\_bus$: $LDWmessage$

8. $CAN\_bus \rightarrow CSC\_ECU$: $LDWmessage$

9. $CSC\_ECU \rightarrow CAN\_bus$: $priorityCSC$

10.||9. $HU\_ECU \rightarrow CAN\_bus$: $priorityHU$

11. $CAN\_bus \rightarrow CSC\_ECU$: $busAssigned$

12.||11. $CAN\_bus \rightarrow HU\_ECU$: $busBusy$

A sequence diagram-based possible trace of the CAN bus access is presented in Figure 29.

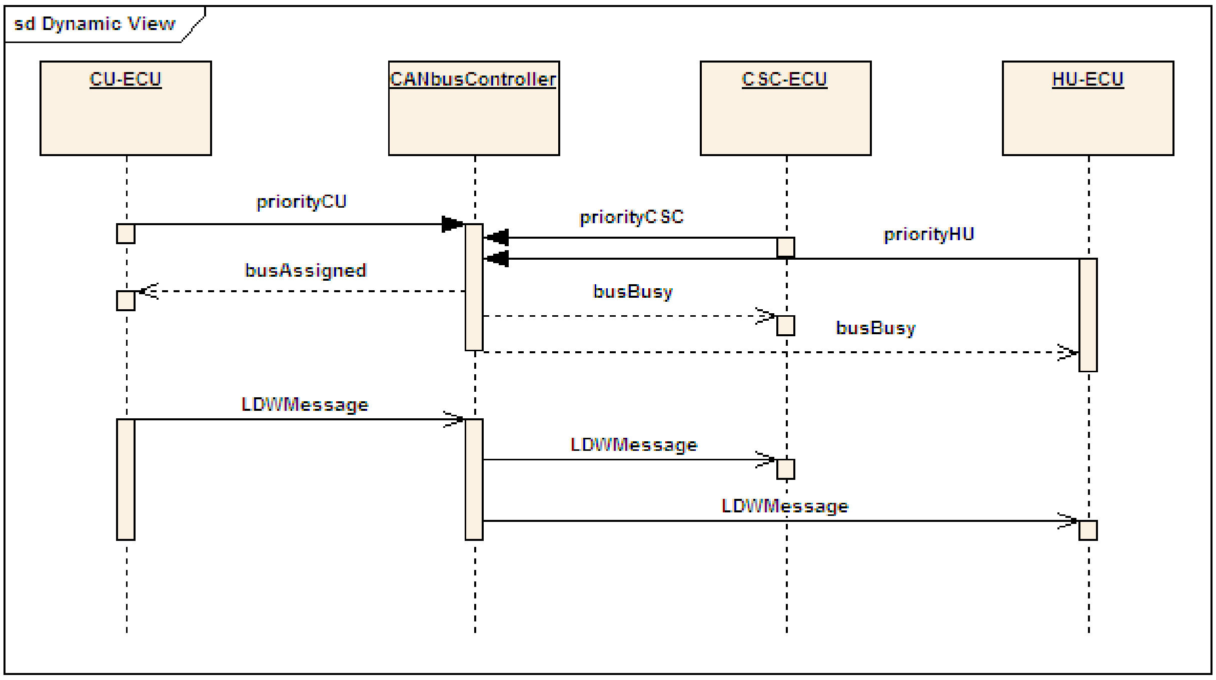


Figure 29 ECUs accessing the CAN bus

4.1.2 Target of Verification

Since we aim to represent and analyze protocols from a magnified approach, we introduce the idea of a Target of Verification (ToV). A Target of Verification is defined through the specification of the following elements:

1. A use case (UC) or an excerpt of a use case (see [33]).

2. EVITA hardware/software modules or EVITA components involved in the use case (see [64]).

3. An Attack Tree (AT) node related to the proposed use case and infrastructure. A specific attack is thus addressed (see [55]).

4. A set of Security Requirement Diagrams (see [55], Appendix D.2.3)

5. A set of protocols are defined in the infrastructure. Several communications can be thus included in the model (see [19]).

The instance that will be modeled in TURTLE is presented in Table 4.

Use Case: UC.2 Local Danger Warning from other cars. Cooperative Awareness Message (CAM) is received in *CU_ECU* and a LDW is sent to *HU_ECU* and *CSC_ECU* (Steps 2–7 from UC description, [33], page 19)
HW/SW: *CU_ECU*: *full featured EVITA HSM. HU_ECU*: *full featured EVITA HSM. CSC_ECU*: *medium featured EVITA HSM.* (see [64], pages 29–35)
Attack Tree: AT.5 *Tamper with Warning Message.* Sub-tree root node: *[5.3] Display incorrect warning message.* (see [55], pages 72 and 74)
Security Requirements: FSR-1.1.2 *Message Freshness along functional Path.* (see [55], page 126)
Protocols: 1) CAM-LDW Protocol. 2) CAN bus access. (Already described in this chapter)

Table 4 Target of Verification for protocol representation.

Even if our integrated modeling and verification approach allows for a wide variety of analyzes and test, we are mainly focused on the security effectiveness of the protocol. As it is stated in the ToV definition, the protocol should satisfy *Freshness Security Requirement*. An important assumption to evaluate the effectiveness is that such security requirement relies on the correct identification of the involved elements, including agents, initial knowledge and assumptions. For attacker representation, we consider the *Dolev-Yao* threat approach. The Dolev-Yao threat model considers that an attacker can overhear, intercept, and synthesize messages in the system in such way that the attacker is only limited by the constraints of cryptographic methods.

4.2 Modeling and Simulating Protocols in TURTLE

TURTLE stands for Timed UML and RT-LOTOS Environment and is a UML profile that allows standardized analysis and design for software and hardware systems with temporal constraints. A TURTLE design is composed by class diagrams which describe the static architecture of the system under design and by activity diagrams which describe the internal behavior of each class. Hence, architecture components (software and hardware) can be abstracted as classes. Signals as well as data interchanges between those components find a suitable representation through the available set of operators which allow synchronization, paralleling, sequencing, invocation and preemption of model instances. Analogous operations can be performed at Activity Diagram level. As it is a real-time modeling profile, a TURTLE design may contain time variables and operators thus allowing model simulation and evaluation.

TTool fully implements this profile and allows automatic model translation into different formal representations. For real time systems the natural candidate for translation is the *Real Time Logic of Temporal Order Specification* (RT-LOTOS) Language [13]. Such formal representation permits model validation either by formal and exhaustive verification or by simulation of specific scenarios.

4.2.1 Model of the ToV

The TURTLE design profile is used to model the integrated ToV. The TURTLE design is composed of a set of classes and objects, thus allowing to design a system at different levels of abstractions. Each class/object in the model is defined through a list of attributes, a list of communication ports – called gates – and one Activity Diagram that defined the behavior of the class/object. Interconnections between classes can be made through several types of operators [3]. TTool includes a TURTLE design editor and TURTLE-to-formal specifications code generators accessible with a press-button approach.

In the next paragraphs, we briefly describe the classes as well as the data structures used to model this ToV.

PDU: This data structure is used to represent messages in the system. It includes the following fields: *origin*, two possible *targets*, the *priority* of the message, a *timestamp* field as well as ten *data* subfields.

TamperedAttackerVehicle: This class represents a tampered external vehicle that uses its *Communications Unit* to send forged messages to another (non tampered) vehicle. The CAM is periodically sent through Dedicated Short-Range Communications (DSRC) system. The periodicity as well as the maximum number of CAMs can be set in the activity diagram. These parameters can optionally be stored as class attributes. The lower bound for the periodicity is 1 clock unit. The first four data of the Protocol Data Unit (PDU) are used to represent the CAM, the time stamp and the corresponding signature of these data with $Pseudo-PrK$. The next data fields are used to transport the certificate that includes $Pseudo-PuK$ and is signed with the private key of the CA $CA-PrK$.

CU_ECU: The *Communications Unit* (CU) in the main ECU receives the CAMs. First the integrity and authenticity of the certificate are validated. Afterwards the integrity, authenticity and freshness of the CAM are verified. The freshness requirement is validated according to the *actualTime*, the *timeStamp* of the message as well as the *timeWindowCU* interval. *timeWindowCU* is a time interval that establishes a criterion to discard delayed messages. After security checks the *CU_ECU* generates a MAC message using the LDW contained in the CAM. The MAC of the LDW includes the *timeStamp* and is generated with the preshared $SesK$.

CU_CRS: This class represents the CRS module of the Communications Unit. It simulates encryption and decryption functionalities in such way that signatures and MACs can be performed in a suitable way. The time cost for these operations is stored in the parameter *cypherDelay*.

CAN_bus: The model of the CAN bus includes a bus assignation scheme based on message *Priority*. When different modules need to transmit a message through the CAN bus, their priorities are first sent. Then, the bus transfers the message with the highest priority. Modules with rejected requests try to retransmit after a given delay. This delay is stored in *waitCANavailability*. Also, the delay induced by CAN bus operations is modeled with the *CANdelay* variable. At last, the model of the CAN bus only considers the access to the bus, as described in the previous sections.

UTClock: A Universal Time Clock (UTC) for the system is modeled with this class. This UTC periodically increases a variable that represents the *actualTime* of the system. *UTClock* is invoked each time that a module in the system asks for the current time (*actualTime*).

CANFreshnessObserver: This module verifies that the messages being sent through the *CAN_bus* truly satisfy freshness criterion. *FreshnessVerifier* uses its own time window (*timeWindowFV*) to verify freshness satisfaction. This module is therefore relevant to check freshness along functional path (the targeted requirement). *CANFreshnessObserver* is a modeling instance that is external to the system implementation. However such an observer assists verification tasks. We assume that this observer is able to collect all the frames of a certain message and is able to verify the time stamp with respect to the global time. The behavior of this instance doesn't alter the inherent properties of the system model.

CSC_ECU: The Chassis and Safety Controller (CSC) receives the LDW from the CU module. Through the MAC, the CSC module validates message integrity and authenticity. Afterwards message freshness is verified based on a time window called *timeWindowCSC*. If the message satisfies security requirements, an algorithm is started in order to attend LDW message. Otherwise a freshness-failure signal is received and the message is discarded. The aforementioned CSC algorithm is not included in this model.

CSC_CRS: The CRS module of the CSC unit. It simulates encryption and decryption tasks whose computation delays are stored in *cypherDelay*.

HU_ECU: The Head Unit receives a copy of the LDW message. Once the message is received, integrity and authenticity are verified. Then, message freshness is checked. In case the message is valid, then a LDW alert is displayed in the HMI. Otherwise the message is discarded and no alert is displayed.

HU_CRS: The Cryptographic Services Module (CRS) of the HU controller. It simulates encryption and decryption tasks whose computation delays are stored in *cypherDelay*.

Parameters: This data structure stores the global parameters of the system. It includes modules identification, symmetric and asymmetric keys, tasks delay as well as freshness time intervals assigned to each module. These *Parameters* play an important role in the model operation and verification since they can be used to explore scenarios in which replay attacks are possible, and also to tune the model in order to prevent such attacks.

Figure 30 represents an overview of the class diagram. In this figure the elements afore described are shown, as well as the operators were used to relate them.

In Figure 31, we present how the access to the CAN bus is modeled as an activity diagram linked to the CAN_bus class.

In the proposed model, we show how several protocols instances can be abstracted and integrated. In the case of CAM-LDW, the protocol is represented through several classes and operators which determine communication channels. Such model integrates the access to the *CAN_bus* as the inner activity diagram of the respective class. The model takes into account component interaction from different infrastructure layers. Since the ToV definition addresses a given requirement, and one related attack, that requirement satisfiability is modeled through the fact that this attack can be successful or not. Verification results, based on simulation and formal verification techniques, are presented in next subsections.

4.2.2 Simulation Results in TURTLE

In previous sections, we have described how an UML-based profile can be used in order to represent/model protocols. We also gave some insights about criteria for protocol evaluation. So far, we presented the interaction between the protocol to distribute a Cooperative Awareness Message (CAM) in the in-car infrastructure and the access to the CAN bus. This approach was based on the TURTLE profile (Section 4.2). In the next subsections, we provide results using simulation and formal verification techniques applied to the protocol models.

4.2.3 Simulation of TURTLE models using RTL

The results presented in this section are related to the protocol model presented in Section 4.2.

Once TTool has checked for the syntax of a TURTLE model, TTool can automatically generate a corresponding RT-LOTOS formal specification. Simulation can then be performed for a given *execution time* of the model, on the RT-LOTOS specification using the RTL toolkit [54]. RTL simulator makes a differentiation between *logical* and *temporal* transitions of the model. The *logical actions* correspond to actions on synchronization gates or internal non-synchronized gates. *Temporal actions* increase the system global time. RTL offers the possibility to assign priorities either to logical actions or to temporal actions with the following options:

logical_actions > temporal_actions: The logical actions have greater priority with respect to temporal actions, thus all pending logical actions are executed before the temporal ones.

logical_actions = temporal_actions: Logical and temporal actions have the same priority.

The space of possible simulation traces therefore depends on the selected option.

The simulation traces can be presented in chronological order, thus describing actions in a logical sequence, or in timing order, thus representing actions in a discrete clock

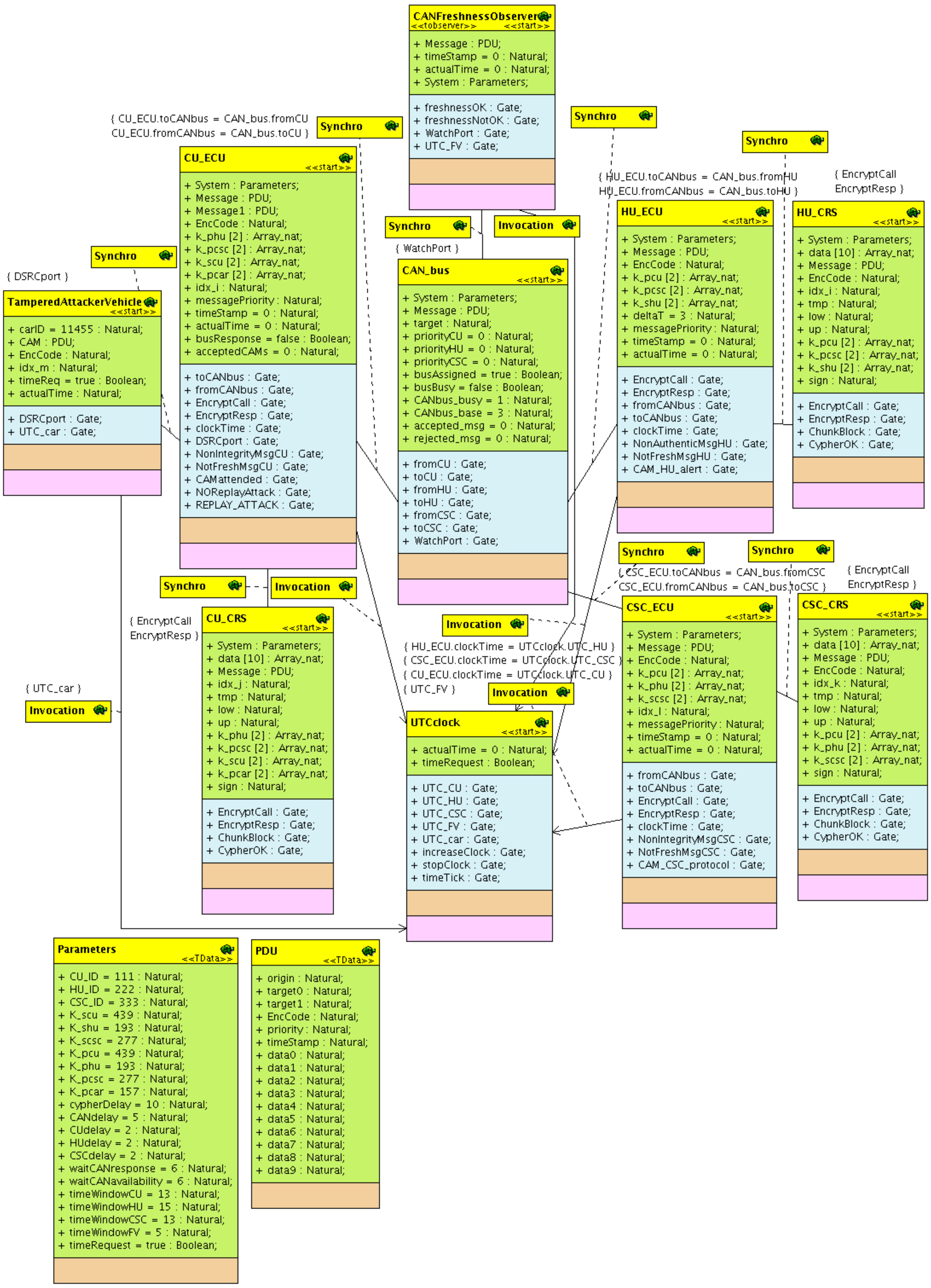
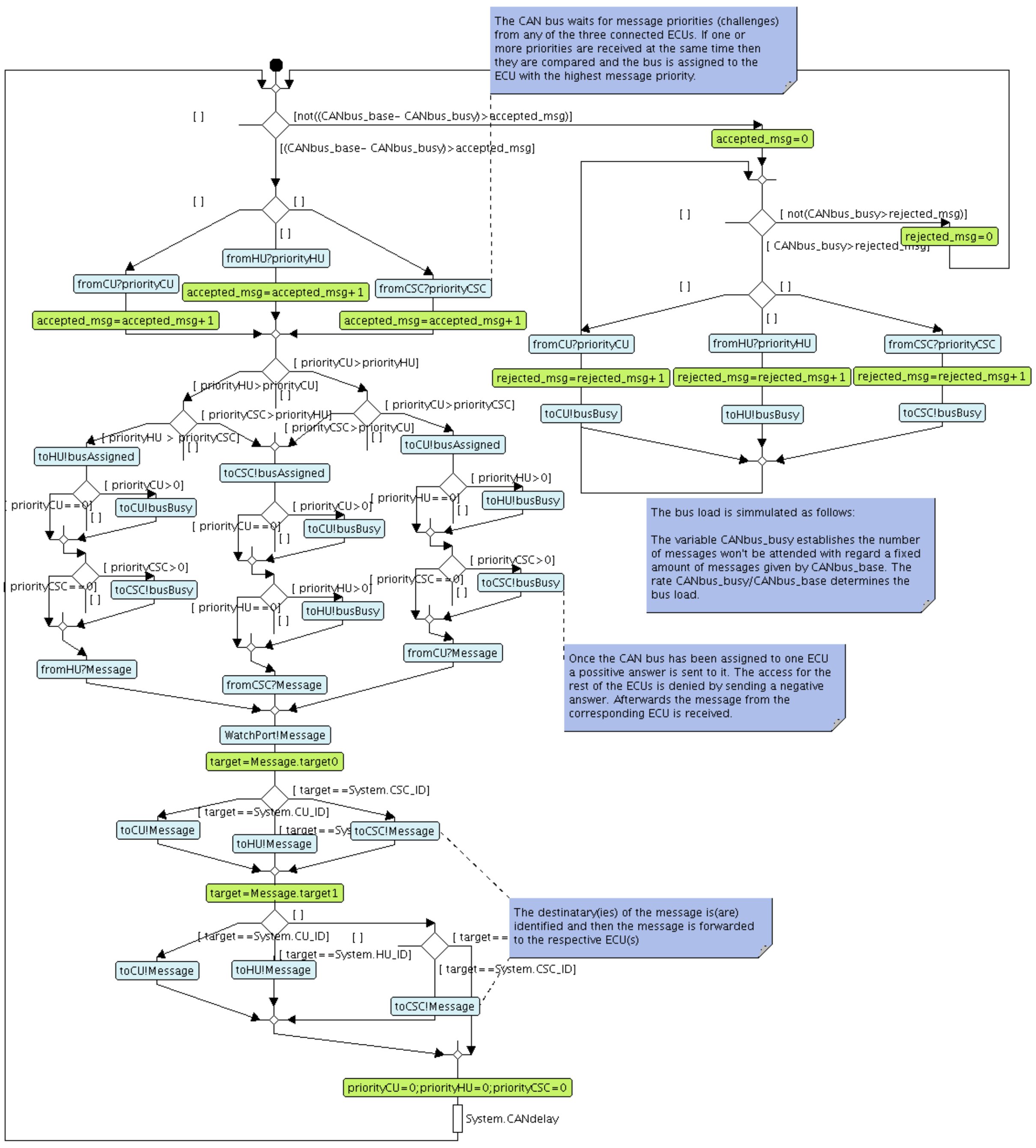


Figure 30 Class diagram of the ToV model



The CAN bus waits for message priorities (challenges) from any of the three connected ECUs. If one or more priorities are received at the same time then they are compared and the bus is assigned to the ECU with the highest message priority.

The bus load is simmulated as follows:

The variable CANbus_busy establishes the number of messages won't be attended with regard a fixed amount of messages given by CANbus_base. The rate CANbus_busy/CANbus_base determines the bus load.

Once the CAN bus has been assigned to one ECU a possitive answer is sent to it. The access for the rest of the ECUs is denied by sending a negative answer. Afterwards the message from the corresponding ECU is received.

The destinatary(ies) of the message is(are) identified and then the message is forwarded to the respective ECU(s)

Figure 31 An overview of the CAN bus protocol diagram

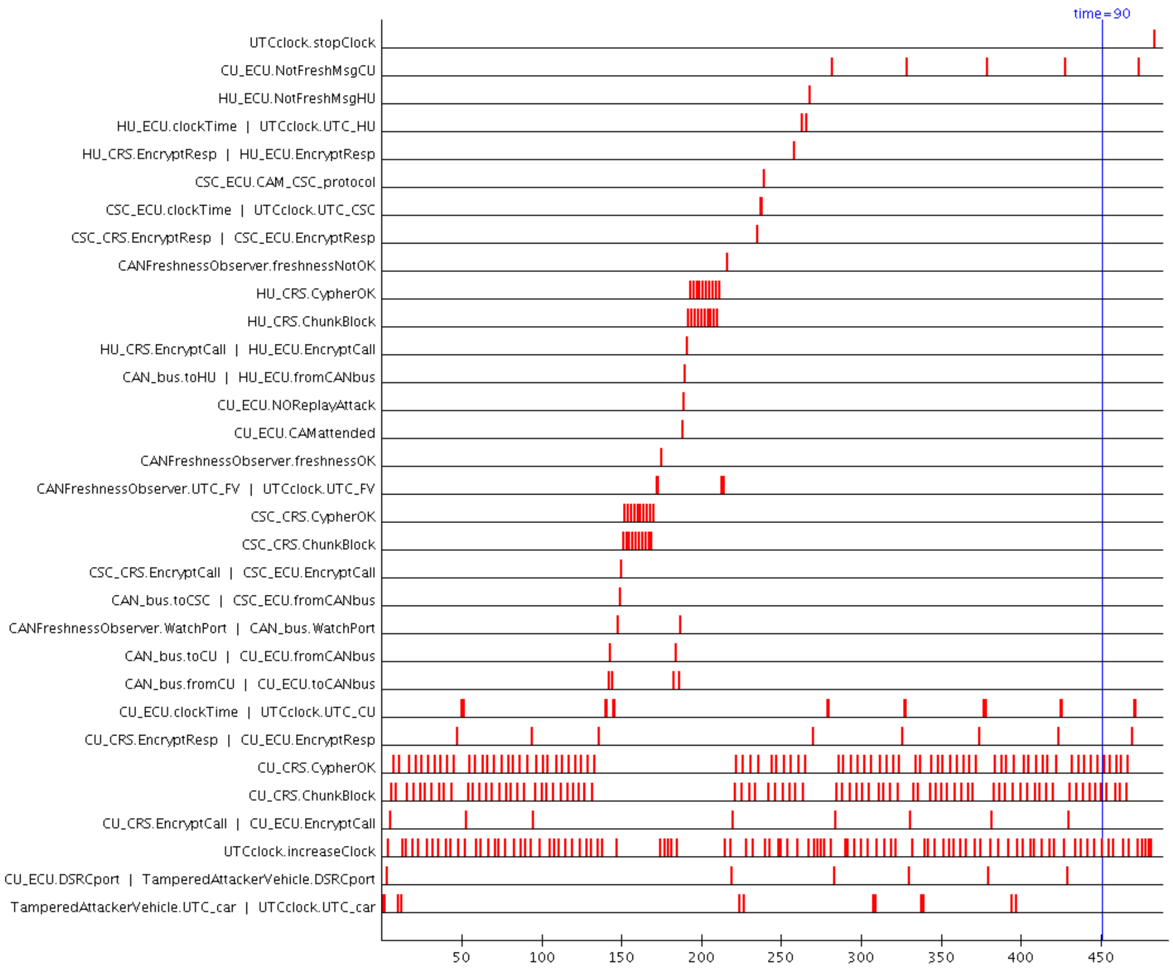


Figure 32 Protocol simulation trace in chronological order using RTL

time scale. The actions represented with small vertical marks in the simulation trace correspond to synchronizations on gates i.e., to data exchange between system entities, or to internal actions performed by entities. Figure 32 presents the view of a simulation trace in chronological order. The gates that were used for at least one synchronization or internal action during the simulation time are shown in the vertical scale. The horizontal scale shows the chronological order of actions.

In Figure 33, we can observe a simulation trace in timing order. Contrary to the previous simulation, the horizontal scale is a time line. Each interval in this scale represents a clock unit. The gates that were used at the same clock time are drawn at the same horizontal index. As a result, two or more actions on the same gate during the same clock time are represented with a single vertical mark.

The evaluation of the protocol traces can be thus based on the following functional and security aspects:

Non-Synchronized gates: The behavior of non-synchronized gates (or in other words, an internal action) can be analyzed in the simulation trace. For instance it can be verified that the gate *UTClock.increaseTime* is used every time unit.

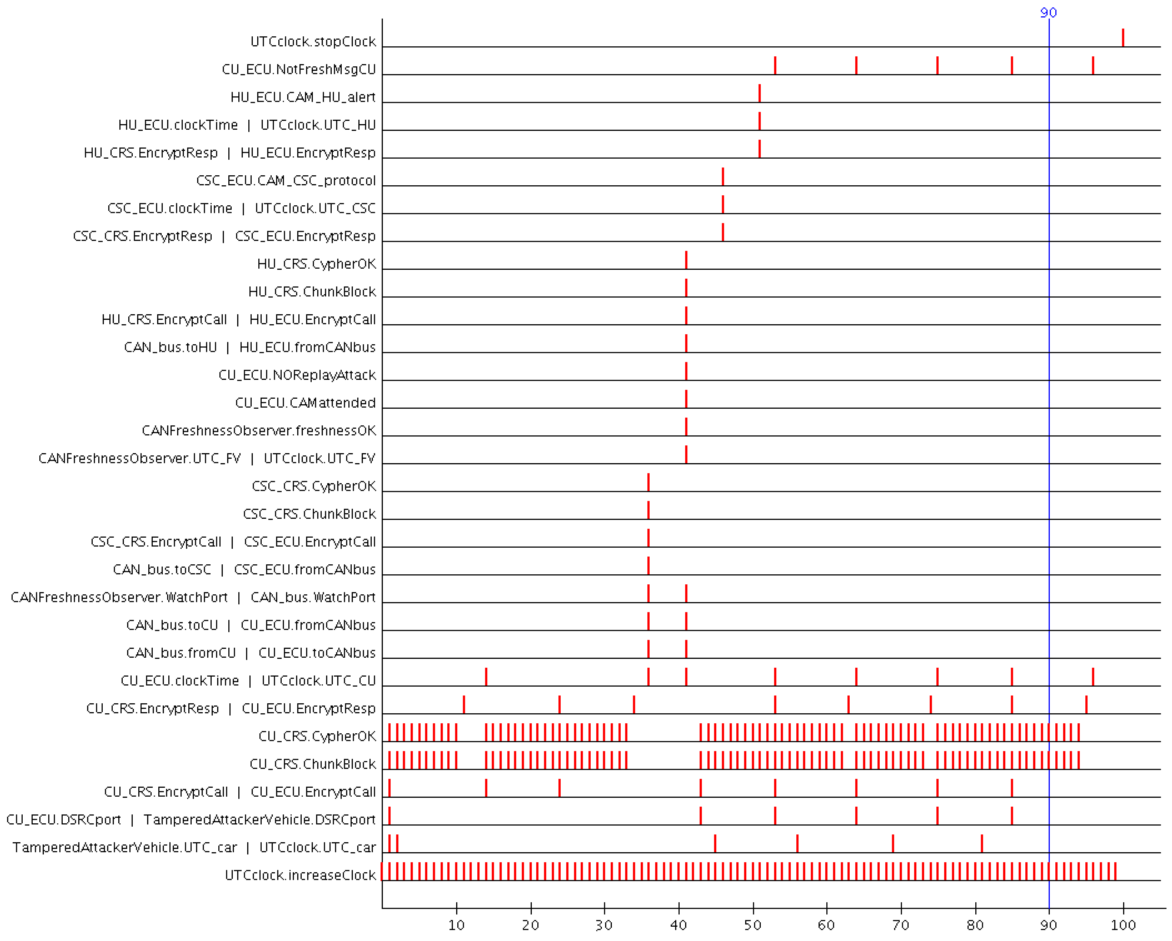


Figure 33 Protocol simulation trace in timing order

Synchronized gates: The behavior of synchronized gates can also be explored. For instance, it can be verified that whenever a message is sent from the Communications Unit (CU_ECU) to the CAN bus (CAN_bus), the two synchronized gates $CAN\_bus.fromCU$ and $CU\_ECU.toCANbus$ are activated.

Invocation calls: These gates are bidirectional thus allowing *send* and *receive* operations in the same port. An example of such invocation is the Communications Unit that requests and retrieves the time through the invocation gate $CU\_ECU.clockTime\ |UTClock.UTC\_CU$.

Activation delay: The difference between the action time on a gate and the time at which the expected response is obtained – with another gate – can be measured. Such response delay may help to analyze time constraints in the protocol. An example of such a delay is the one provoked by the CAN bus protocol. Such a delay can be measured between the gates $CAN\_bus.fromCU|CU\_ECU.toCANbus$ and $CAN\_bus.toCSC|CSC\_ECU.fromCANbus$.

Security gates: Every entity in the model is assigned with specific gates on which an action is performed whenever a security property is either violated or satisfied thus allowing explicit model verification of security properties at simulation time. Examples of such security gates are $CU\_ECU.NotFreshMsgCU$ and $CANFreshness Observer.freshnessOK$.

4.2.4 Simulation of TURTLE models using UPPAAL

UPPAAL was jointly developed by Uppsala University, Sweden and Aalborg University, Denmark. UPPAAL is both the name of the language, based on communicating timed automata [2], and a toolkit with modeling, simulation and model-checking capabilities. UPPAAL has already been successfully used to formally model and verify systems [47], including protocols [39].

To overcome limitations of RTL – which offers no model-checking capabilities – TTool can automatically generate UPPAAL specifications from the TURTLE designs. The UPPAAL specification keeps a quite intuitive relationship with the original TURTLE model thus providing more simplicity when simulation or verification are made. In a first stage each Tclass of the TURTLE model is translated to a finite timed automaton which is labeled with the same name as the Tclass. Each automaton is composed by:

Vertices or nodes: A vertex or node represents a state in the automaton. Vertices can be labeled with **O** to indicate an *initial* state, with **C** to indicate a *committed* state or **U** to indicate a *urgent* state. A *committed* state indicates that this state must be left without any other transitions taken in another automata. A *urgent* state must be left before the time elapses.

Edges: An edge represents a directed transition between two vertices. Each transition can be assigned/defined with one or more of the following elements:

1. *Guard:* A guard is a time or data constraint. Time constraints are of the form $C \triangleright n$ where C is a variable in the system that measures the time and is called Clock, n is an integer value and $\triangleright \in \{<, =, >, \geq, \leq\}$. Time constraints of the form $C_1 - C_2 \triangleright n$ are also guards. Data constraints have a similar syntax. If the guard is satisfied then the transition can be realized immediately, except if the transition has a synchronization action.
2. *Synchronization actions:* Synchronization actions keep the form *channel!*, which denotes a sending component and *channel?*, which denotes a receiving component. Synchronized actions allow interchanges between automata. The absence of synchronization labels on an edge implies an internal non-synchronized transition path.
3. *Urgent Channels:* gates of TURTLE models are translated to UPPAAL Urgent channels. An urgent channel defines a synchronized transition that is carried out without the passage of time.
4. *Clock and variable settings*

Figure 34 represents the timed automaton of the *FreshnessVerifier* class as generated by TTool from the TURTLE model.

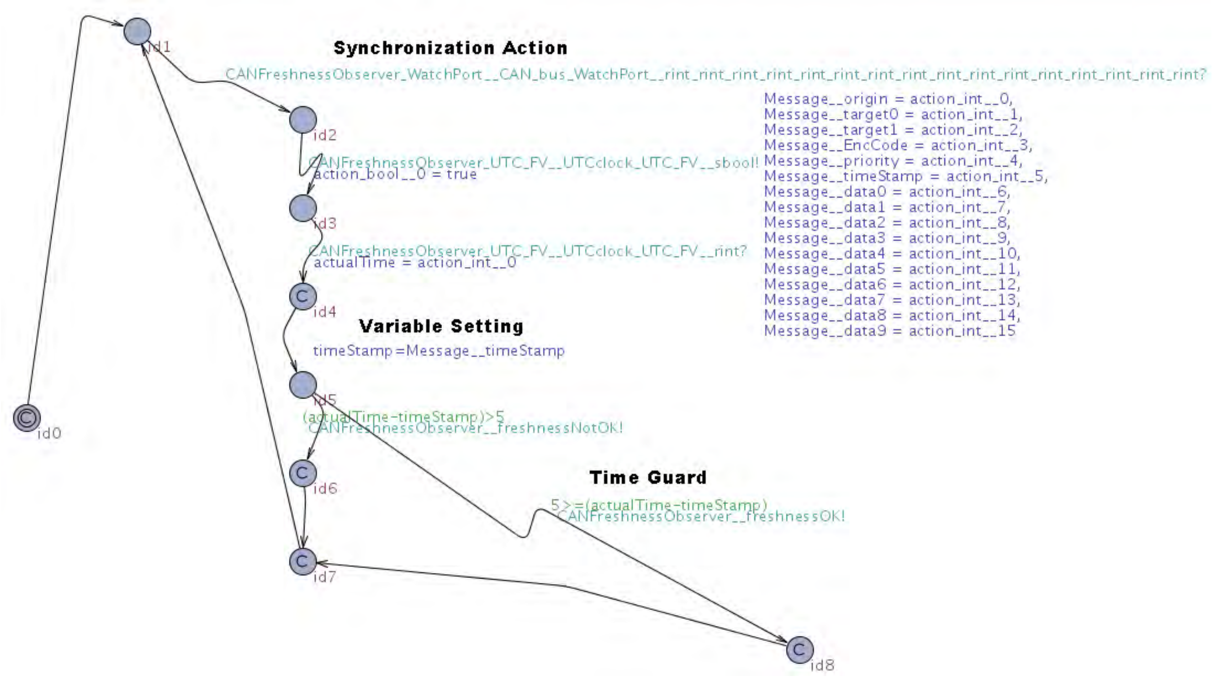

id1
Synchronization Action
CANFreshnessObserver_WatchPort__CAN_bus_WatchPort__rint_rint_rint_rint_rint_rint_rint_rint_rint_rint_rint_rint_rint_rint_rint_rint?
Message__origin = action_int__0,
Message__target0 = action_int__1,
Message__target1 = action_int__2,
Message__EncCode = action_int__3,
Message__priority = action_int__4,
Message__timeStamp = action_int__5,
Message__data0 = action_int__6,
Message__data1 = action_int__7,
Message__data2 = action_int__8,
Message__data3 = action_int__9,
Message__data4 = action_int__10,
Message__data5 = action_int__11,
Message__data6 = action_int__12,
Message__data7 = action_int__13,
Message__data8 = action_int__14,
Message__data9 = action_int__15
id2
CANFreshnessObserver_UTC_FV__UTCclock_UTC_FV__sbool!
action_bool__0 = true
id3
CANFreshnessObserver_UTC_FV__UTCclock_UTC_FV__rint?
actualTime = action_int__0
id4
Variable Setting
timeStamp=Message__timeStamp
id5
(actualTime-timeStamp)>5
CANFreshnessObserver__freshnessNotOK!
id0
id6
Time Guard
5>=(actualTime-timeStamp)
CANFreshnessObserver__freshnessOK!
id7
id8

Figure 34 Timed automaton in UPPAAL of the FreshnessVerifier class

The UPPAAL simulator provides an interactive panel which generates simulation traces in the form of sequence diagrams. The simulation trace starts with the initial vertice of each automaton and is depicted as a tuple of nodes: ($id0, id9, id31, id55, id438, id464,$ $id530, id556, id662, id1045,$ $id1539$) (see Figure 35).

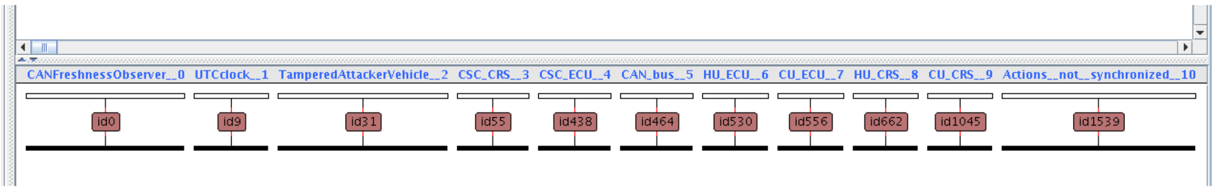
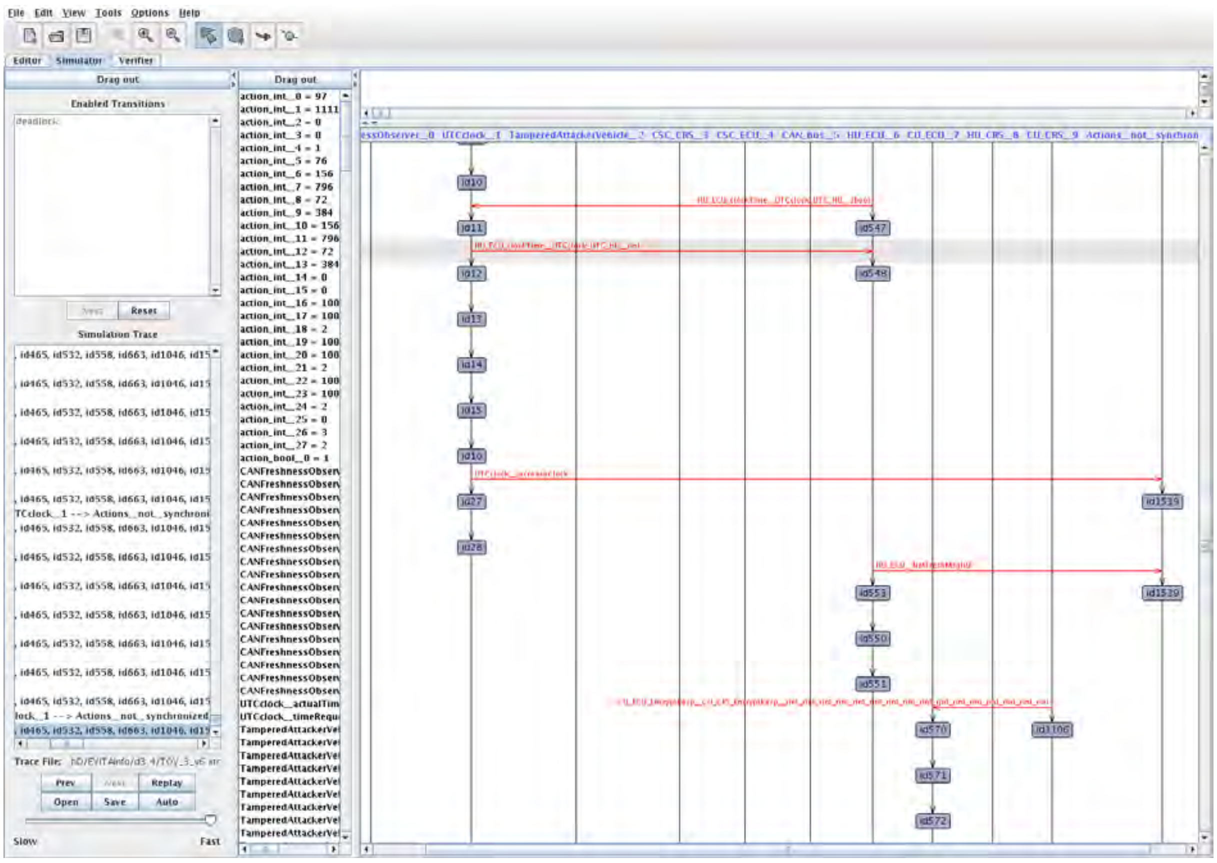


Figure 35 Initial status of the simulation trace

The lifeline of each automaton reflects any transition from an origin state to a destination state (see Figure 36). Moreover, if an action in a channel is performed, a message is displayed from the origin automata to the destination automata. Such interactions are related with information interchanges of our protocol. At last, simulation states can be saved for further use.

UPPAAL also offers verification capabilities. *UPPAAL verifier* is a model checker taking as input a UPPAAL specification and a set of CTL formulae (called *queries*). A query can be of the form:

- $A[]\phi$: ϕ is always satisfied.
- $E[]\phi$: ϕ exists.
- $A <> \phi$: ϕ is always eventually satisfied.


File Edit View Tools Options Help
Editor Simulator Verifier
Drag out
Enabled Transitions
deadlock
Next Reset
Simulation Trace
Drag out
Prev Next Replay
Open Save Auto
Slow
Fast
essObserver__0 UTCclock__1 TamperedAttackerVehicle__2 CSC_CRS__3 CSC_ECU__4 CAN_bus__5 HU_ECU__6 CU_ECU__7 HU_CRS__8 CU_CRS__9 Actions_not_synchron
id10
id11
id12
id13
id14
id15
id16
id27
id28
id547
id548
id553
id550
id551
id570
id571
id572
id1519
id1520
id1106

```
, id465, id532, id558, id663, id1046, id15
, id465, id532, id558, id663, id1046, id15
, id465, id532, id558, id663, id1046, id15
, id465, id532, id558, id663, id1046, id15
, id465, id532, id558, id663, id1046, id15
, id465, id532, id558, id663, id1046, id15
TCclock__1 --> Actions__not__synchroni
, id465, id532, id558, id663, id1046, id15
, id465, id532, id558, id663, id1046, id15
, id465, id532, id558, id663, id1046, id15
, id465, id532, id558, id663, id1046, id15
, id465, id532, id558, id663, id1046, id15
, id465, id532, id558, id663, id1046, id15
, id465, id532, id558, id663, id1046, id15
lock__1 --> Actions__not__synchronized
, id465, id532, id558, id663, id1046, id15
Trace File: hD/EVITAinfo/d3.4/TOV_3_v6.xtr
```

```
action_int__0 = 97
action_int__1 = 1111
action_int__2 = 0
action_int__3 = 0
action_int__4 = 1
action_int__5 = 76
action_int__6 = 156
action_int__7 = 796
action_int__8 = 72
action_int__9 = 384
action_int__10 = 156
action_int__11 = 796
action_int__12 = 72
action_int__13 = 384
action_int__14 = 0
action_int__15 = 0
action_int__16 = 100
action_int__17 = 100
action_int__18 = 2
action_int__19 = 100
action_int__20 = 100
action_int__21 = 2
action_int__22 = 100
action_int__23 = 100
action_int__24 = 2
action_int__25 = 0
action_int__26 = 3
action_int__27 = 2
action_bool__0 = 1
CANFreshnessObserv
CANFreshnessObserv
CANFreshnessObserv
CANFreshnessObserv
CANFreshnessObserv
CANFreshnessObserv
CANFreshnessObserv
CANFreshnessObserv
CANFreshnessObserv
CANFreshnessObserv
CANFreshnessObserv
CANFreshnessObserv
CANFreshnessObserv
CANFreshnessObserv
CANFreshnessObserv
CANFreshnessObserv
CANFreshnessObserv
CANFreshnessObserv
UTCclock__actualTim
UTCclock__timeRequ
TamperedAttackerVe
TamperedAttackerVe
TamperedAttackerVe
TamperedAttackerVe
TamperedAttackerVe
TamperedAttackerVe
TamperedAttackerVe
TamperedAttackerVe
TamperedAttackerVe
```

Figure 36 General overview of a simulation trace

- $E <> \phi$: ϕ eventually exists.
- $\phi_1 \rightarrow \phi_2$: ϕ_1 implies/leads to ϕ_2

In Table 5, we show a list of queries that were used to evaluate freshness along functional path in the model, using reachability and liveness properties, and the result of each query.

Query	**Semantics**	**Result**
$A[]CU\_ECU\_\_7.id575 == 1\ imply\ (CU\_ECU\_\_7.actualTime - CU\_ECU\_\_7.Message\_\_timeStamp) <= 13$	Freshness in CU	Satisfied
$A[]CSC\_ECU\_\_4.id457 == 1\ imply\ (CSC\_ECU\_\_4.actualTime - CSC\_ECU\_\_4.Message\_\_timeStamp) <= 13$	Freshness in CSC	Satisfied
$A[]HU\_ECU\_\_6.id549 == 1\ imply\ (HU\_ECU\_\_6.actualTime - HU\_ECU\_\_6.Message\_\_timeStamp) <= 13$	Freshness in HU	Satisfied
$A[]CANFreshnessObserver\_\_0.id8 == 1\ imply\ (CANFreshnessObserver\_\_0.actualTime - CANFreshnessObserver\_\_0.timeStamp) < 5$	Freshness in CAN bus	Satisfied

Table 5 List of queries to evaluate freshness in the simulation trace.

4.3 Modeling and Simulating Protocols in DIPLODOCUS

In DIPLODOCUS, protocols (Transport, Session, Presentation, Application) can be efficiently modeled by means of one dedicated task per layer. Low level details belonging to the physical layer like pins, signals, modulation, voltages, cable specifications, etc. are completely abstracted. As bus contention is considered as a decisive factor for the performance of the final system, it obviously has to be taken into account. The user can specify the arbitration policy of a bus just by graphically selecting the corresponding parameter (fixed priorities, round robin, CAN, ...).

In addition to media access, merely error detection mechanisms are normally present in data link protocols. In the systems modeled so far, transmission errors, their detection and recovery have not been identified as the main limiting factors. However, the assumption may not hold in electromagnetically polluted environments. In this case, it would be interesting to introduce abstract models of loss and overhead due to error detection and correction, respectively. The network layer is reflected by a static routing procedure which is carried out at model transformation stage (from UML to simulation code) in case the routing has not been defined explicitly by the user. The outcome of the routing procedure is the mapping of an abstract application-level channel on n buses, $n - 1$ bridges and potentially one storage node depending on the associated Central Processing Units (CPUs) of source and destination task and the topology of the network.

In the particular case of the CAN bus model, the following assumptions were made:

1. The protocol is based on fixed priorities assigned to messages (as opposed to bus masters).

2. As the protocol implies a point-to-multipoint communication scheme, a corresponding semantics had to be introduced at application level. An additional operator was added to the DIPLODOCUS environment so as to allow to write samples to several abstract channels at a time. However the strict separation of application and architecture still holds. If the channels referenced by the new operator are mapped onto a single bus supporting multipoint-to-point communication, the data is only sent once. Otherwise, the data is sent once for each channel.

3. The data rate of a CAN bus is taken into account by a parameter which can be set directly using the graphical interface.

4. CAN bus transfers normally do not comprise an external memory – messages to be sent and received are temporarily stored inside bus transceivers. As the inner structure of bus transceivers is not captured by DIPLODOCUS hardware architecture diagrams, the semantics of a mapping merely including buses and bridges had to be defined. Given that a CAN message is only conveyed on the bus once, solely the application-level write command issues a bus transaction, whereas the application-level read command is interpreted as a read access to the internal memory of the local CAN transceiver. Thus, the read command may exhibit different semantics depending on the mapping of the respective channel.

So far, we have declared two metrics characterizing the performance of communication architectures:

- Usage of buses.
- Contention delays

Other metrics could be provided if necessary.

4.3.1 Model of the CAM-LDW protocol in DIPLODOCUS

The specification of the DIPLODOCUS profile considers the "Y-Approach" methodology [40]. This methodology distinguishes between applications and architectures. Such differentiation implies that an explicit mapping from the involved applications onto the related architecture should be performed. Since the applications are considered as a network of concurrent communicating processes, a such mapping associates application processes to specific architecture components. The "Y-Approach" is based on a so called Trace-Driven approach [40] which allows tracing of the assigned workload for each architecture component. Hence, the composed system permits architecture-application-mapping performance evaluation in terms of work load and simulation time. As an heritage from this approach, in the DIPLODOCUS profile two complementary models should be specified; the Application Model and the Architecture Model.

The Application Model is intended to represent the functional specification of the system. Thus functionalities, mechanisms and services which are implemented and executed at different SW/HW layers can be abstracted and represented as *Tasks* in the same diagram. In this paradigm each *Task* is defined through a list of attributes and an Activity Diagram that states its behavior. Interactions between *Tasks* find a suitable representation in one of the three link operators:

Channels: Are a mean to modeling data exchange between *Tasks*.

Events: Are a mean for *Tasks* to interchange signals that may synchronize *Task* execution.

Requests: Are a mean for *Tasks* to produce/trigger other *Tasks*. If the requested *Task* is busy then the *Request* is stored into a FIFO thus waiting for processing.

The Architecture Model is the mean to represent hardware elements of the architecture. For this purpose DIPLODOCUS provide several generic architecture components which contain the notion of time thus allowing model simulation and performance evaluation. The following nodes have been specified in the profile:

1. *CPU*
2. *Hardware Accelerator*
3. *Memory*
4. *Bus*
5. *Bridge*

Architecture nodes are communicated using simple links (arrows) thus reflecting the architecture distribution. At last but not least, a mapping from the Application Model onto the Architecture Model should be performed; every *Task* as well as every *Channel*, *Event* and/or *Request* is thus mapped onto a single architecture node. Since the DIPLODOCUS design can be translated to a SystemC and to a LOTOS specification, simulation as well as static formal analysis can be transparently performed by the designer [4]. In the next paragraphs we give a more detailed description of the Application and Architecture Models for the CAM-LDW protocol.

Application Model: The DIPLODOCUS model of the CAM-LDW protocol contains the following functional entities: The task *OutsideWorld* represents the simulation test bench and generates an initial CAM message. The *ProtocolDispatcher* task receives that message, verifies its certificate and subsequently its signature and afterwards extracts the necessary items to determine and compose the LDW message. The *CheckLDW* task in turn performs a plausibility check of the latter message. In case the message is considered to be significant, the *ProtocolDispatcher* generates a MAC of the LDW message and forwards them to the *SafetyCheck* and to the *Display* tasks. The former is in charge of verifying the validity of the LDW message MAC and applying safety related algorithms which are modeled in terms of their computational cost. The *Display* task just verifies the MAC of the received LDW message and notifies the potential danger to the driver. Three *CRS* tasks, namely CRS_for_PD, CRS_for_SC and CRS_for_Disp, provide cypher related functions to the afore mentioned tasks. Three services have been modeled in terms of their computational complexity and their data transfers: *Encryption*, *certificate verification* and *signature verification* which also contains MAC verification. A general overview of this Application Model is presented in Figure 37. An overview of a representative Activity Diagram in the model is shown in Figure 38.

Architecture model and mapping: The architecture model consists of three ECUs: ECU_CU, ECU_CSC and ECU_HU. All ECUs are connected to the main CAN bus. Every ECU is defined in terms of a System-on-Chip consisting of a local bus, a CPU, a HSM and a memory Random-Access Memory (RAM). The tasks *ProtocolDispatcher* and *LDWApplication* are mapped onto the internal CPU of the Communications Unit. *SafetyCheck* is executed by the CPU of the ECU_CSC and last ECU_HU is loaded with the *Display* task. Security related functions (*CRS* tasks) are deployed on dedicated HSMs. A complementary view of this Architecture Model is presented in Figure 39.

4.3.2 Simulation Results in DIPLODOCUS

Several assumptions have been considered in order to realize model simulation. These assumptions are intended to emulate real size of the data exchanges. However the model admits variable parametrization thus considering future modifications that may bring the model to a better correspondence with respect to real size values.

The main assumptions made are listed bellow. These assumptions impact the simulation process and consequently should be taken into account for results evaluation.

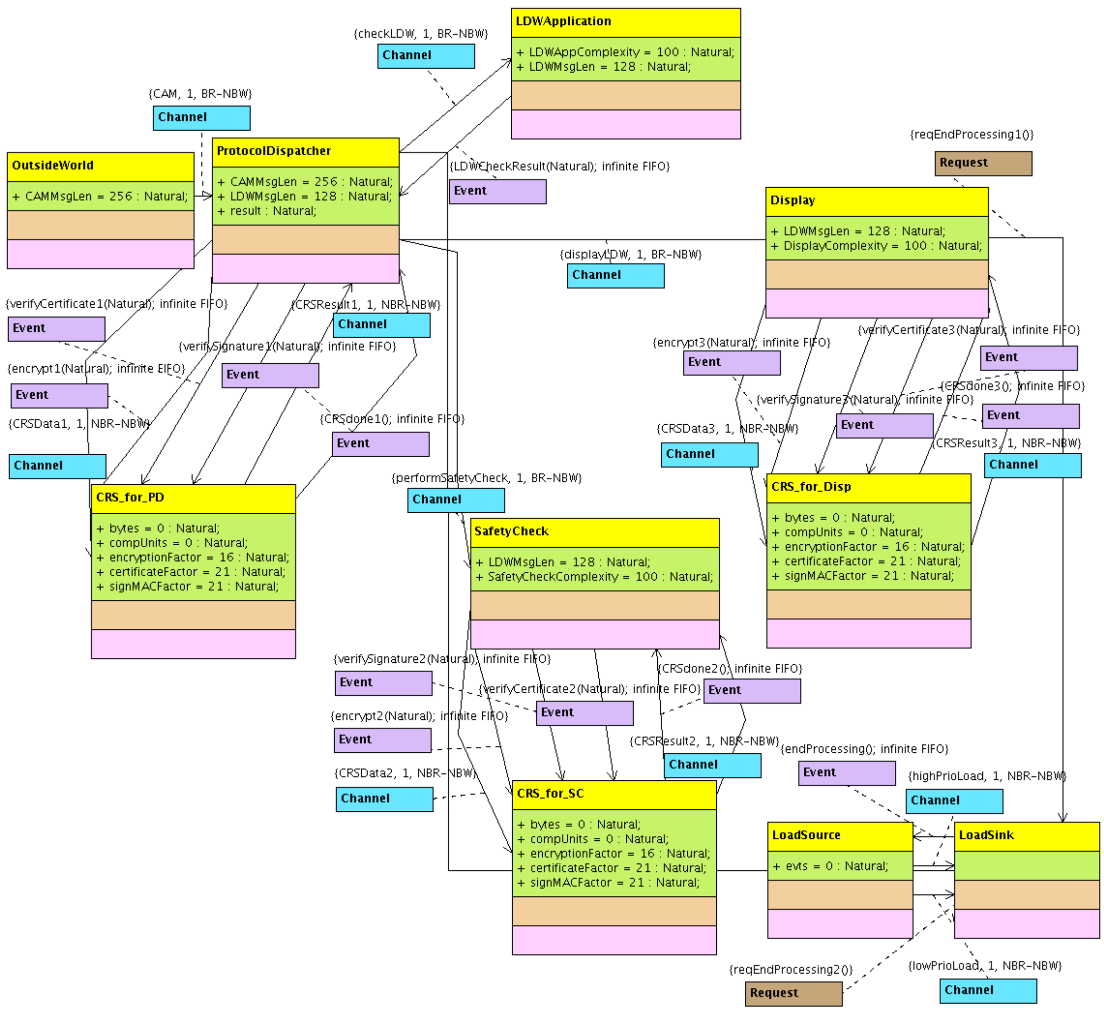

LDWApplication
+ LDWAppComplexity = 100 : Natural;
+ LDWMsgLen = 128 : Natural;
{checkLDW, 1, BR−NBW}
Channel
{CAM, 1, BR−NBW}
Channel
OutsideWorld
+ CAMMsgLen = 256 : Natural;
ProtocolDispatcher
+ CAMMsgLen = 256 : Natural;
+ LDWMsgLen = 128 : Natural;
+ result : Natural;
{LDWcheckResult(Natural); infinite FIFO}
Event
{reqEndProcessing1()}
Request
Display
+ LDWMsgLen = 128 : Natural;
+ DisplayComplexity = 100 : Natural;
{displayLDW, 1, BR−NBW}
Channel
{verifyCertificate1(Natural); infinite FIFO}
Event
{CRSResult1, 1, NBR−NBW}
Channel
{verifySignature1(Natural); infinite FIFO}
Event
{encrypt1(Natural); infinite FIFO}
Event
{CRSData1, 1, NBR−NBW}
Channel
{CRSdone1(); infinite FIFO}
Event
CRS_for_PD
+ bytes = 0 : Natural;
+ compUnits = 0 : Natural;
+ encryptionFactor = 16 : Natural;
+ certificateFactor = 21 : Natural;
+ signMACFactor = 21 : Natural;
{performSafetyCheck, 1, BR−NBW}
Channel
SafetyCheck
+ LDWMsgLen = 128 : Natural;
+ SafetyCheckComplexity = 100 : Natural;
{encrypt3(Natural); infinite FIFO}
Event
{verifyCertificate3(Natural); infinite FIFO}
Event
{CRSdone3(); infinite FIFO}
{verifySignature3(Natural); infinite FIFO}
Event
Event
{CRSData3, 1, NBR−NBW}
Channel
{CRSResult3, 1, NBR−NBW}
Channel
CRS_for_Disp
+ bytes = 0 : Natural;
+ compUnits = 0 : Natural;
+ encryptionFactor = 16 : Natural;
+ certificateFactor = 21 : Natural;
+ signMACFactor = 21 : Natural;
{verifySignature2(Natural); infinite FIFO}
Event
{verifyCertificate2(Natural); infinite FIFO}
Event
{CRSdone2(); infinite FIFO}
Event
{encrypt2(Natural); infinite FIFO}
Event
{CRSResult2, 1, NBR−NBW}
Channel
{endProcessing(); infinite FIFO}
Event
{CRSData2, 1, NBR−NBW}
Channel
{highPrioLoad, 1, NBR−NBW}
Channel
CRS_for_SC
+ bytes = 0 : Natural;
+ compUnits = 0 : Natural;
+ encryptionFactor = 16 : Natural;
+ certificateFactor = 21 : Natural;
+ signMACFactor = 21 : Natural;
LoadSource
+ evts = 0 : Natural;
LoadSink
{reqEndProcessing2()}
Request
{lowPrioLoad, 1, NBR−NBW}
Channel

Figure 37 Application Modeling of the CAM-LDW protocol

1. Size of CAM: 256 bytes.
2. Size of the LDW Message: 128 bytes.
3. The computational complexity of algorithms applied in *LDWApplication*, *Safety Check*, *Display*, has been arbitrarily set to 100 computation units. These values can be refined in order to include more realistic complexities.
4. A message is transferred to the dedicated *CRS* tasks as a whole. Subsequently, the three above mentioned security functions may be applied to that data.
5. The response to a signature/MAC/certificate verification command is small and its impact on the bus load is negligible.
6. For CAM messages the data to be processed in order to verify the certificate amounts to roughly half the message size.
7. For LDW messages the data to be processed in order to verify the MAC amounts to roughly the message size.

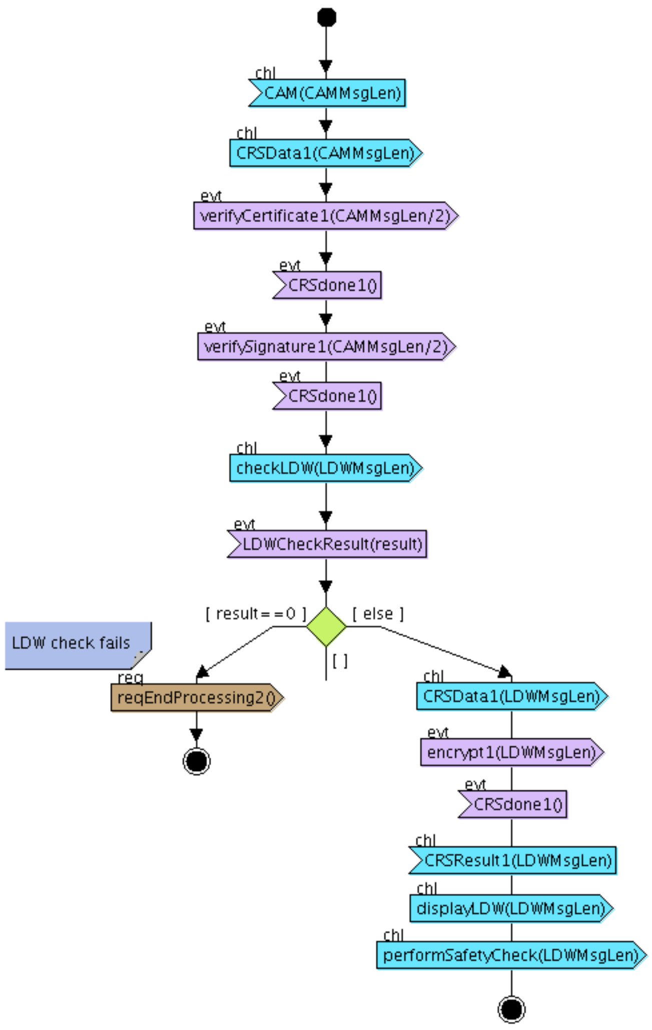


Figure 38 View of *Protocol Dispatcher Task* Activity Diagram

8. The computational cost for encryption, certificate verification and signature/MAC verification is a linear function of the message size. The corresponding factors called *encryptionFactor*, *certificateFactor* and *signMACFactor* (contained in the *SRC* Tasks) can be modified to reflect more realistic factors.

9. The main CAN bus is constantly loaded at 33% by dummy tasks (*LoadSource*, *LoadSink*). This traffic accounts for functionalities which have not been modeled.

The model is simulated by generation, compilation and execution of the corresponding SystemC specification. The user is assisted by the graphical interface of the DIPLODOCUS simulator in order to accomplish the simulation process. An overview of the graphical interface is presented in Figure 40.

The DIPLODOCUS framework currently provides two main measures to evaluate the integrated model; indeed Contention Delay and Load. The Contention Delay provides the number of clock cycles that an architecture node is delayed before accessing the communication media. As well the Load provides a percentage of the computing complexity was demanded for task execution. In Table 6 the loads and contention delays for all the

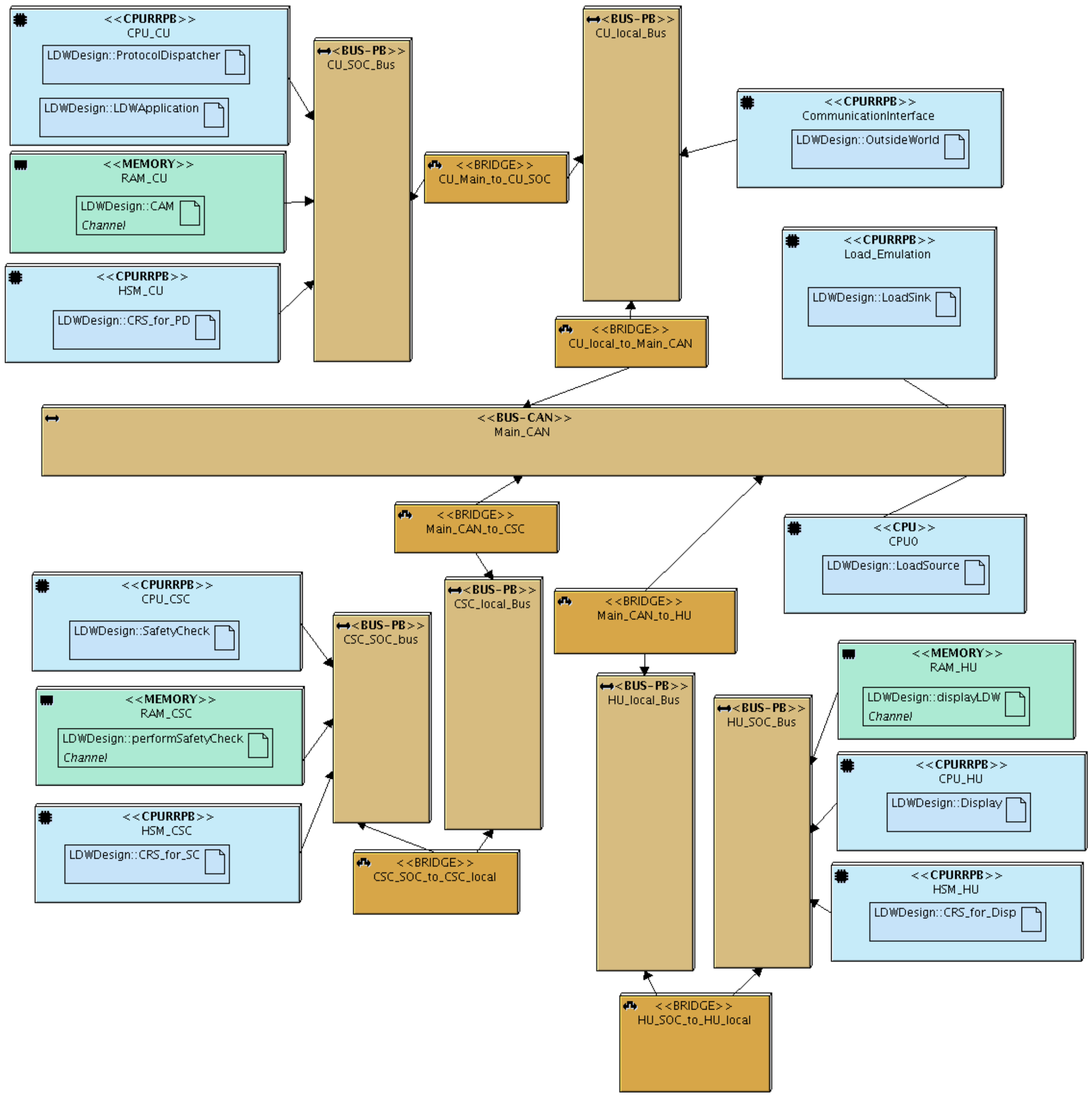
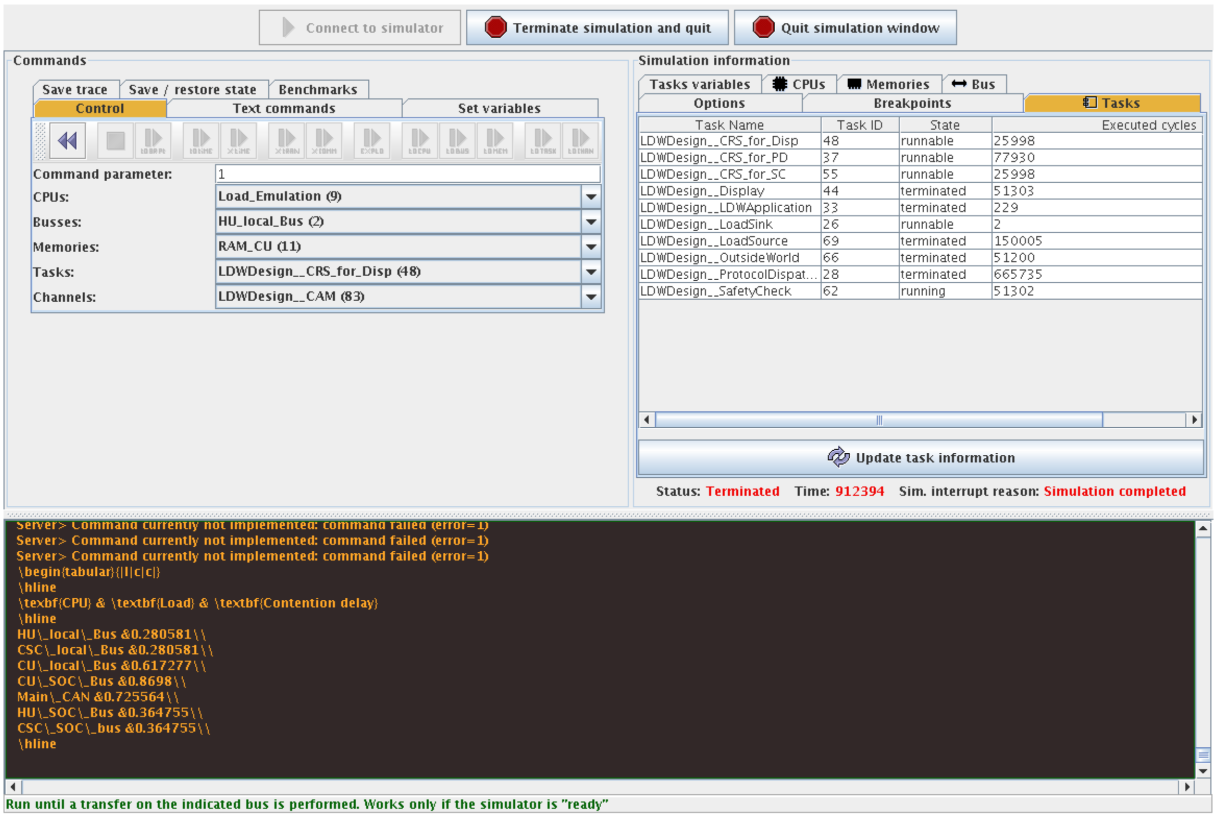


Figure 39 Overview of the Architecture Model for the CAM-LDW protocol simulation. This view includes the respective mapping of the Application Model.

computing elements are shown. In Table 7 we present the impact of the protocol execution in the load of the local buses (internal ECU SoC buses) as well as in the load of the main CAN bus.



Command parameter: 1
CPUs: Load_Emulation (9)
Busses: HU_local_Bus (2)
Memories: RAM_CU (11)
Tasks: LDWDesign__CRS_for_Disp (48)
Channels: LDWDesign__CAM (83)

Task Name	Task ID	State	Executed cycles
LDWDesign__CRS_for_Disp	48	runnable	25998
LDWDesign__CRS_for_PD	37	runnable	77930
LDWDesign__CRS_for_SC	55	runnable	25998
LDWDesign__Display	44	terminated	51303
LDWDesign__LDWApplication	33	terminated	229
LDWDesign__LoadSink	26	runnable	2
LDWDesign__LoadSource	69	terminated	150005
LDWDesign__OutsideWorld	66	terminated	51200
LDWDesign__ProtocolDispat...	28	terminated	665735
LDWDesign__SafetyCheck	62	running	51302

Status: Terminated Time: 912394 Sim. interrupt reason: Simulation completed

```
Server> Command currently not implemented: command failed (error=1)
Server> Command currently not implemented: command failed (error=1)
Server> Command currently not implemented: command failed (error=1)
\begin{tabular}{|l|c|c|}
\hline
\texbf{CPU} & \textbf{Load} & \textbf{Contention delay}
\hline
HU\_local\_Bus &0.280581\\
CSC\_local\_Bus &0.280581\\
CU\_local\_Bus &0.617277\\
CU\_SOC\_Bus &0.8698\\
Main\_CAN &0.725564\\
HU\_SOC\_Bus &0.364755\\
CSC\_SOC\_bus &0.364755\\
\hline
```

Run until a transfer on the indicated bus is performed. Works only if the simulator is "ready"

Figure 40 Front-end of the DIPLODOCUS simulator

CPU	**Load**	**Contention delay**
Load_Emulation	–	–
CPU_CU	0.729908	5085.71
HSM_CU	0.0854127	0
CPU_HU	0.056229	0
HSM_HU	0.0284943	0
HSM_CSC	0.0284943	0
CPU_CSC	0.0562279	0
CommunicationInterface	0.0561161	6666.67
CPU0	0.164408	25672.7

Table 6 Computing node loads and contention delays

CPU	**Load**
HU_local_Bus	0.280581
CSC_local_Bus	0.280581
CU_local_Bus	0.617277
CU_SOC_Bus	0.8698
Main_CAN	0.725564
HU_SOC_Bus	0.364755
CSC_SOC_bus	0.364755

Table 7 Local buses and main bus loads for protocol simulation

5 Conclusions

This document specifies a set of protocols that are applicable within automotive on-board architectures in order to secure dedicated applications and scenarios. The protocols integrate with the EVITA security architecture presented within [64], hence using the functionality of the EVITA HSMs and the abstractions provided by the security software framework.

Notably, we have designed key management protocols for efficiently initializing and updating keys inside the vehicle amongst HSMs. Based on the EVITA security architecture, we have instantiated a KeyMaster entity, which resides on a dedicated ECU or may be replicated, depending on the on-board architecture. The design of the key management protocols considers the features provided by the EVITA HSMs. Key management is a crucial task in on-board networks, as a cost-effective solution needs to be provided that can handle off-line and on-line key management scenarios at the same time. The concept of a KeyMaster is specifically important. We differentiate three types of HSMs, where the EVITA Light HSM does only support symmetric cryptography, hence suitable communication protocols are needed to securely deploy keys on the Light HSMs and to establish communication with other units equipped with Light, Medium and Full HSMs. Besides that, keying protocols for group communication and asymmetric encryption have been proposed.

Additionally, we have proposed a protocol for efficiently issuing and exchanging attestations (trust statements) within a vehicular on-board network, which are issued or enforced by a respective entity, e.g., the security controller in a virtualized environment. The proposed concept is also applicable for V2X communication scenarios, where attestations are exchanged between vehicles. The security controller operates in a trusted isolated environment and enforces respective access policies. It is already feasible to deploy our approach in on-board architectures, where only one multipurpose ECU is available since it enables the secure integration of less trusted applications with trusted environments.

In order to provide security over various bus systems, we have specified a security transport protocol, which defines a security payload and is exemplarily mapped to the transport protocol CTP, which can be used over CAN and FlexRay. Depending on the application scenario, the security payload may be integrated directly with a respective bus communication protocol, e.g. CAN, FlexRay, or MOST.

The protocols build the foundation for reaching the security requirements identified for the respective use cases. We have exemplified how the specified protocols can be used for dedicated application scenarios. These application scenarios comprise secure flashing, maintenance scenarios, secure device integration and local danger warning applications.

In addition, based on the LDW scenario, we exemplarily show how these protocols can be analyzed with respect to performance and security based on TTool. Further analysis and formal verifications is outlined within [16].

A Related Work

A.1 SeVeCom

A.1.1 SeVeCom objectives

SeVeCom [37, 57] addresses the security of future vehicle communication networks, including both the security and privacy of V2X communication. Its objective is to define the security architecture of such networks, as well as to propose a roadmap for progressive deployment of security functions in these networks.

V2X communications bring the promise of improved road safety and optimized road traffic through co-operative systems applications. To this end a number of initiatives have been launched, such as the Car-2-Car consortium in Europe, and the DSRC in North America. A prerequisite for the successful deployment of vehicular communications is to make them secure. For example, it is essential to make sure that life-critical information cannot be modified by an attacker; it should also protect as far as possible the privacy of the drivers and passengers. SeVeCom will focus on communications specific to road traffic. This includes messages related to traffic information, anonymous safety-related messages, and liability-related messages.

SeVeCom specifies an architecture and security mechanisms which provide the right level of protection. It addresses issues such as the apparent contradiction between liability and privacy, or the extent to which a vehicle can check the consistency of claims made by other vehicles. The following topics have been fully addressed: key and identity management, secure communication protocols (including secure routing), tamper proof device and decision on cryptosystem, and privacy.

The definition of cryptographic primitives takes into account the specific operational environment. The challenge was to address (1) the variety of threats, (2) the sporadic connectivity created by moving vehicles and the resulting real-time constraints, and (3) the low-cost requirements of embedded systems in vehicles. These primitives are adaptations of existing cryptosystems to the V2X environment.

A.1.2 Security Mechanisms/Concepts

A big number of security concepts have been studied and taken into account in SeVeCom. Some of them are briefly described below.

Identification & Authentication Concepts The first block of security concepts deals with identity authentication. In contrast to the classical understanding, where authentication means entity authentication, in Vehicular Ad-Hoc Networks (VANETs) we require different forms of authentication, e.g. authentication of vehicle positions or various other attributes. The individual mechanisms are: *Identification, authentication of sender, authentication of receiver, attribute authentication and authentication of intermediate nodes.*

Privacy Concepts In the requirements engineering process a strong need for privacy-preserving mechanisms has been clearly identified. Without that, location tracking and other forms of privacy invasions may seriously damage the adoption of V2X communication systems. Such system must support the following mechanisms: *Total anonymity, resolvable anonymity and location obfuscation.*

Integrity Concepts Ensuring the integrity of communicated information is of vital importance e.g. for all eSafety applications. Modification attacks may render warning information useless or even lead to accidents provoked by the attacker. As communication happens ad-hoc between arbitrary communication partners, not all information can be protected by the classical cryptographic means. Different means of integrity protection are provided like: *Integrity protection, encryption, detection of protocol violation, consistency/context checking, attestation of sensor data, location verification, tamper-resistant communication system, digital rights management and replay protection.*

Access Control/Authorization Concepts Controlling who can participate in various aspects of V2X communications and who can access different parts of the in-vehicle systems will provide additional security.

A.1.3 Overall description

In this section we describe the mechanisms implemented by SeVeCom in order to take into account the concepts above.

Total anonymity Communication is the main cause of privacy problems in V2X communication systems. By communicating, vehicles report their existence plus potentially other data, like their position, speed, but maybe even license plate or owner information. Privacy enhancing mechanisms need to be designed and used in a way that they try to reveal this information from eavesdroppers. Whereas some information may be hidden from eavesdroppers by means of encryption, at least legitimate recipients of data need the ability to decrypt the data. Even worse, the pure existence and identification of a vehicle cannot easily be concealed and will be reported in clear. Therefore, it is envisioned that cars will change their identification regularly by using pseudonyms. Changing pseudonyms may, however, create problems for applications that need a session semantics lasting longer than the interval between pseudonym changes. Here, some session management will be necessary. The pseudonyms can then be used with the authentication mechanisms instead of the regular identifiers. A pseudonym is the couple formed by a key and a short-term certificate and is changed frequently.

Integrity protection Integrity protection is of huge relevance to practically all V2X communication applications as modified data may damage the operation of every application. Therefore, communicated data needs some integrity protection which can be achieved either by

- MACs when using symmetric cryptography or
- digital signatures with asymmetric cryptography.

Problems are that MACs require the exchange of a shared key between sender and receiver and digital signatures may create a huge overhead in terms of bandwidth and computing power. Within SeVeCom the second solution has been chosen: Every message is signed using asymmetric keys and certificates that are not linked to the vehicle.

Encryption Encryption of communicated data is the standard way of ensuring confidentiality in unicast communication and will also be used in VANETs. However, it might be a problem to negotiate session keys in one-way communication. Relying only on asymmetric cryptography on the other hand creates a significant overhead. Group communication and message dissemination create additional problems; however, the analysis has shown that applications using such communication patterns usually need no confidentiality and thus no encryption of data. When the encrypted data also includes checksums and is carefully combined with authentication mechanisms, encryption may also provide data integrity properties. MACs reach a similar goal with symmetric cryptography.

This solution provides controlled access to the network (all messages must be signed) and protects users privacy.

A.1.4 Communication Patterns identified by SeVeCom

Rather than specifying concrete communication protocols, SeVeCom considered the use of different communication patterns in order to be implementation-independent and to facilitate necessary updates of security mechanisms:

- Beaconing: Beacons are packets that are sent via a single hop without being relayed or forwarded by the receiving node. They are typically sent to several receivers, but all of them are supposed to be direct neighbors within a limited area.

- Geobroadcasting: Geobroadcast messages are also unidirectional, but they are distributed via multiple hops and intended for a larger destination region than beacons.

- Geographic Routing: The concept of geographic routing (also called unicast routing) denotes a multiple hop, single path forwarding of a message from a sending node to a receiving node via several intermediate nodes. Regarding external V2X communication, geographic routing may be applied to the dissemination of warning messages. However, we need to make restrictions concerning the general assumptions: It only makes sense to apply this sort of communication patterns if we can assume a “linear sequence” of the vehicles intended for receiving the messages.

- Information Dissemination: The purpose of information dissemination is to store information and to later forward it to further entities.

- Information Aggregation: The purpose of information aggregation is similar as for information dissemination. As additional feature, a reduction of possible overhead for widespread information is provided.

A.1.5 Implementation

The SeVeCom solution deals with the signature and verification of vehicle to vehicle messages. All messages that cross the SeVeCom communication stack are rerouted to the security module of SeVeCom, which undertakes the appropriate actions. The security module is a plugin of the overall communication stack as depicted in Figure 41.

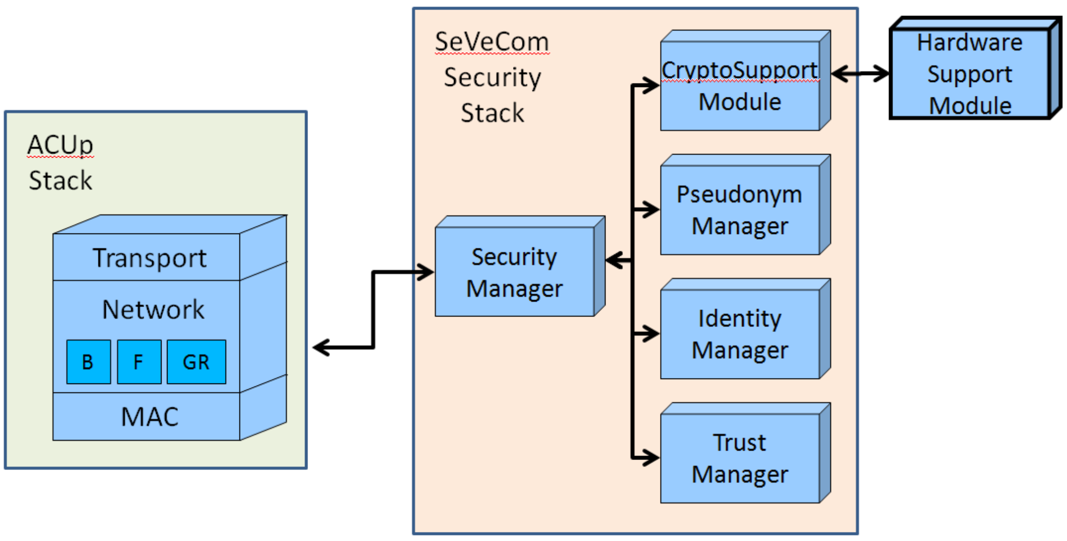


Figure 41 The Security Module is linked to the Communication Stack

A mechanism called the Inter-Layer Proxy (ILP) allows a module (in our case, the SeVeCom security module) to be plugged into the communication stack. For this to work, the ILP offers a register function. Using this interface, the Security Manager tells the ILP what callback (or method) to call when a message of a given type for a given direction is received by the communication stack. The registration is made by the Security Manager during its initialization phase. A configuration file contains all the needed information for this registration (i.e., what method to register for what type of message and for which direction). For example in Figure 42 we can see that the method `send_packet` of the SeVeComSecureBeaconing component is registered for outgoing (= direction) beaconing (= type) messages (step 2). Therefore when such a message is received by the communication stack it is routed to this method (step 2'). This method in turn calls the method `CreateSignature` of the Security Manager. The signed message is then returned to the ILP, which sends it to the lower layers of the communication stack.

The ILPs can be implemented at any level of the communication stack, as well as several ILPs may be present in the stack depending of the needs. The number of ILPs and where to place them is under the responsibility of the communication stack implementer. To select the right position for the ILP in a stack, the following elements should be kept in mind as the Security Stack adds an overhead to the frames:

- If the ILP is too low in the stack, some problems can arise. If the ILP is under the layer doing fragmentation, it means that each fragment will transport an overhead. This will cause very low network performances. If the ILP is too low on a

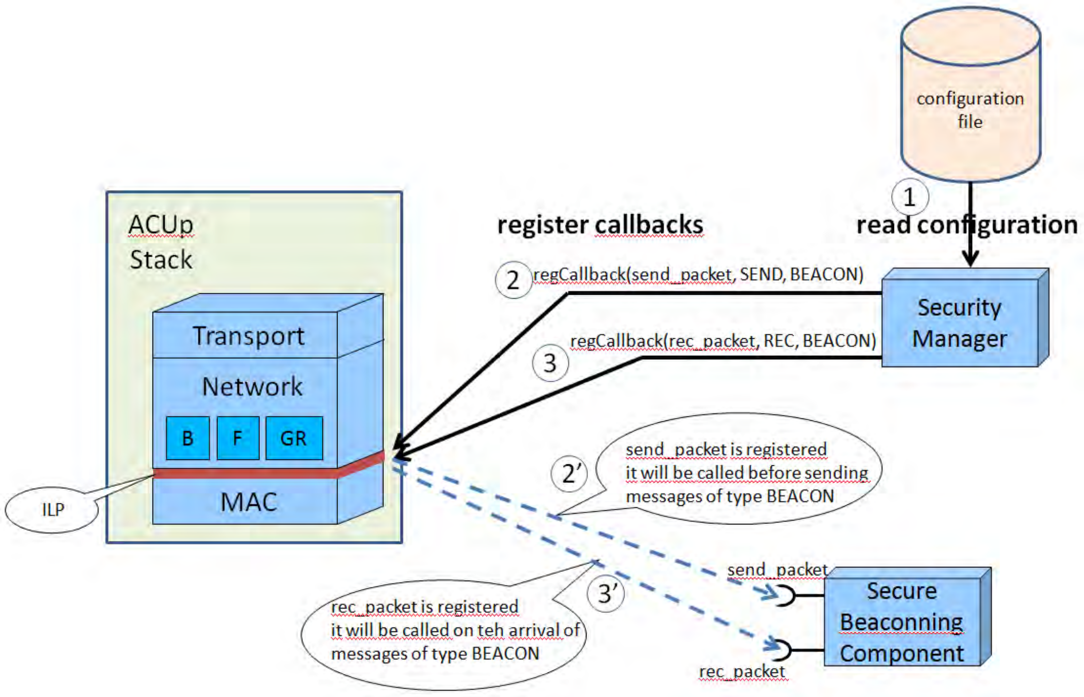

configuration file
1
read configuration
Security Manager
register callbacks
2 regCallback(send_packet, SEND, BEACON)
3 regCallback(rec_packet, REC, BEACON)
ACUp Stack
Transport
Network
B
F
GR
MAC
ILP
2' send_packet is registered it will be called before sending messages of type BEACON
3' rec_packet is registered it will be called on teh arrival of messages of type BEACON
send_packet
rec_packet
Secure Beaconning Component

Figure 42 Registration of a callback by the Security Manager

small-packet-based stack, this overhead may be over the maximum frame size at PHY/Media Access Control (MAC) level.

- If the ILP is too high in the stack, it reduces its usefulness. For example if an in-coming message needs to be rerouted, its hop-counter must be decremented and its signature must be recalculated. This cannot be done if the ILP is above the corresponding layer. In any case the ILP must be low enough to prevent hackers to take advantage of the situation.

As a conclusion, on most stacks, the best place for the ILP is between the MAC layer and the Transport Layer.

Changing a pseudonym certificate is inefficient if the network ids are not changed. For example, if it is an Ethernet/IP stack, the communicating node can be tracked by its MAC address, its IP address and its pseudonym. If only the pseudonym is changed, the MAC address and IP address may still be tracked. The network ids change must be done for every layers in the stack.

Therefore the communication stack must offer another function that changes the MAC address of the wireless device. This function is called by the Security Module each time a new pseudonym is used.

Each outgoing message is passed to the Security Module which adds a time-stamp and the pseudonym certificate and signs it with the pseudonym private key. In order to meet the requirement "extremely fast signature generation" Elliptic Curve Cryptography (ECC) has been chosen. For ECC the signature size is shorter compared to Rivest, Shamir, and Adleman Public Key Encryption Technology (RSA) or Digital Signature Algorithm (DSA) signatures. For privacy purpose, each time the pseudonym is changed, the MAC address of the wireless device must be changed.

Each in-coming message is passed to the Security Module. Its signature is verified, as well as the certificate chain linking the pseudonym certificate to one of the trusted root certificate. Its time-stamp is verified too. If the verification fails, the message is dropped.

The management of certificates (retrieve public certificates, get new pseudonyms, revocation, and so on) if not fully implemented.

A.1.6 Applicability to EVITA

The SeVeCom project has especially covered the topic of e-safety as considered by the EVITA use cases 1, 2, 4, 5 and 8 (as defined in [33]). Regarding message exchange, the EVITA use cases are characterized by warning messages created by one vehicle, transferred to other vehicles and then further processed by ECUs inside the receiving vehicles. Having introduced the SeVeCom architecture and mechanisms in the previous sections, we want to show how to make use of the features provided by the SeVeCom concept within EVITA.

The ILP mechanism described in A.1.5 allows EVITA to use the SeVeCom Security Module. An integration is rather straight forward, because of the plugin-mechanism. All that needs to be done in the host communication stack is to implement two functions for:

- The registration of the security callbacks,
- The changing of the MAC address after pseudonym changes.

When a message of the specified type is received, the registered callback must be called by the host communication stack. This has been fully explained in [37].

In the following listing, the interface that allows for an integration is shown.

```

/**
 * This is the header file that defines the C interface of the
   SevecomProduct library
 */
typedef int SecurityManagerHandle;

/**
 * SecurityManager Constructor
 * @return security manager handle or -1 (error)
 */
SecurityManagerHandle initSecurityManager();

/**
* Ask the SecurityManager to treat a received pdu
* @param handle      [in] security manager handle
* @param receivedPduData     [in] received pdu data (with signature, etc.)
* @param receivedPduDatalength     [in] length of the received pdu data
* @param treatedPduData    [out] treated pdu data (signature verified, etc.)
* @param treatedPduDataLength    [out] length of the treated pdu data
* @return success(1) / fail (0)
*/
int treatReceivedFrame(const SecurityManagerHandle handle, const unsigned
    char* receivedPduData, const unsigned int receivedPduDataLength,
    unsigned char** treatedPduData, unsigned int* treatedPduDataLength);
```

```

/**
* Ask the SecurityManager to treat a sending pdu
* @param handle      [in] security manager handle
* @param sendingPduData      [in] sending pdu data
* @param sendingPduDatalength    [in] length of the received pdu data
* @param treatedPduData    [out] treated pdu data (signature added, etc.)
* @param treatedPduDataLength    [out] length of the treated pdu data
* @return success(1) / fail (0)
*/
int treatSendingFrame(const SecurityManagerHandle handle, const unsigned
    char* sendingPduData, const unsigned int sendingPduDataLength, unsigned
    char** treatedPduData, unsigned int* treatedPduDataLength);

/**
* Indicate to the SecurityManager the result of the network id change
* @param handle         [in] security manager handle
* @param result           [in] id change success (1) / fail (0)
*/
void indicateNetworkIdChange(const SecurityManagerHandle handle, int result
    );

/**
* Register the ILP requestNetworkIdChange function, that allows the
    SecurityManager to request a network id change to the ILP
* @param handle         [in] security manager handle
* @param *requestNetworkIdChange   [in] function pointer for
    requestNetworkIdChange
*/
void registerRequestNetworkIdChange(const SecurityManagerHandle handle, int
     (*requestNetworkIdChange)());

/**
 * SecurityManager Destructor
 * @param handle          [int] security manager handle
 */
void freeSecurityManager(const SecurityManagerHandle handle);
```

A.2 The e-safety application eCall

The term eCall refers to an interoperable in-vehicle emergency call service which, in case of a crash, automatically calls the nearest emergency center. Crash information is sent together with current GPS location, vehicle travel direction, vehicle identification data, and time of the event to the so-called Public Service Answering Point (PSAP) as shown in Figure 43. The eCall use case is composed of six different domains as shown in Figure 44.

1. In case of an accident, e.g. detected by the trigger of the airbag, an emergency call is automatically generated.
2. The eCall generators initiate the eCall by sensors triggered and/or manually, send the in-vehicle information to a PSAP. The in-vehicle care information consists of last positions of the vehicle (position chain) based on GPS/Galileo signals, vehicle identity.

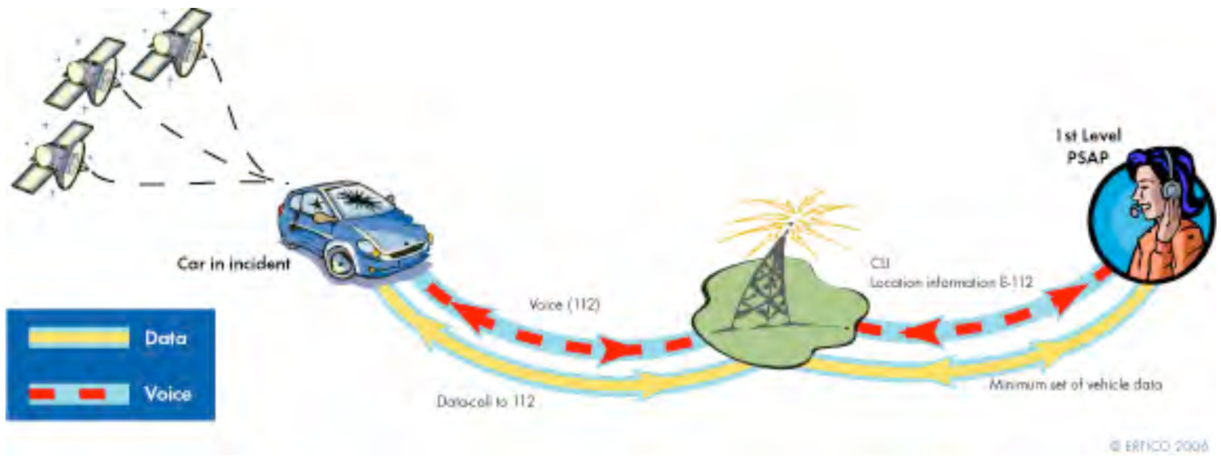


Figure 43 The eCall System

3. The eCall carried through the mobile network, is recognized by the Mobile Network Operator (MNO) as a 112 emergency call.

4. Based on the 112 handling the MNO enriches the call with the Caller Line Identification (CLI), and at the same time, according to the USD and the E112 recommendation, add the best location available.

5. After the 112 handling, the MNO delivers the 112-voice together with the CLI, mobile location and the eCall MSD to the appropriate PSAP.

6. The PSAP transmits an acknowledgment to the eCall generator specifying that the MSD have been properly received.

A.2.1 eCall: In-Band modem solution

In order to facilitate and guarantee roaming, common standardized interfaces and data transfer protocols are needed. As an action to support a full pan-European service the European Commission has requested ETSI to standardize the eCall interface between the eCall generator and the PSAP along with the transport protocol. Different technical possibilities for the transmission of emergency data have been studied by GSM Europe (GSME) and by ETSI MSG. The conclusion was that only an in-band modem can provide a secure solution that maximizes the service coverage area, offers a prioritized and fast transmission of the Minimum Set of Data (MSD), and minimizes the required modifications to the cellular core networks and the public switched telephone network (PSTN). The European Commission and ETSI MSG have delegated 3rd Generation Partnership Project (3GPP) to standardize an in-band modem specification for eCall. The main design objective for the eCall in-band modems is transmission speed and reliability. During the transmission of an MSD, the voice communication is muted, and the duration of muting should be minimized in an emergency situation. On the other hand, it is also essential that the MSD is transmitted in an extremely reliable way, mandating the use of a CRC controlled ARQ scheme. The requirements on transmission speed and reliability have

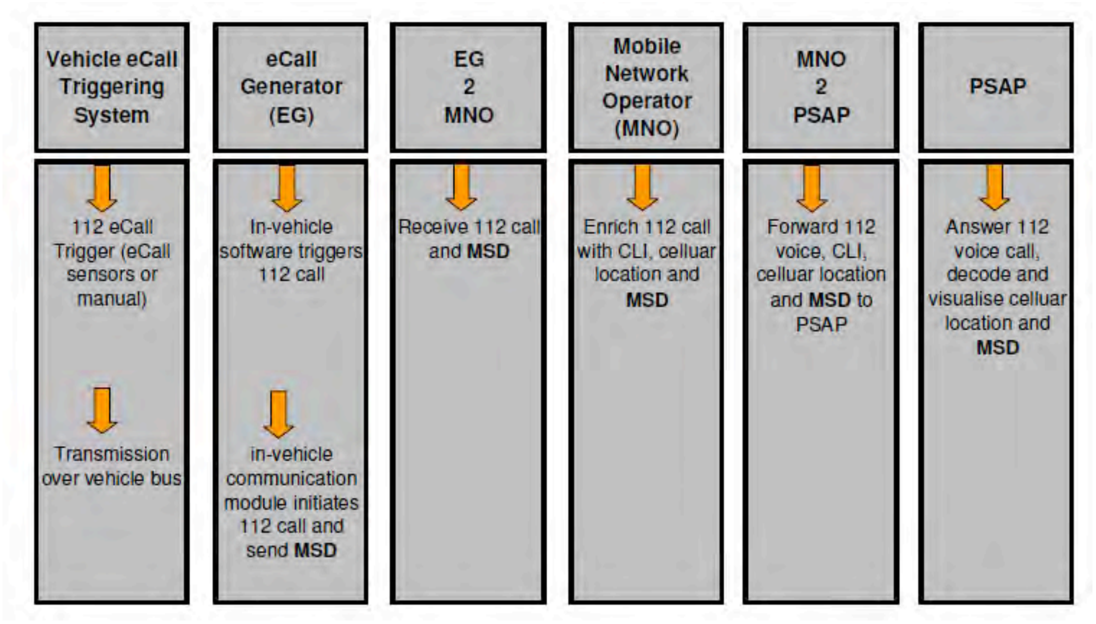

Vehicle eCall Triggering System
eCall Generator (EG)
EG 2 MNO
Mobile Network Operator (MNO)
MNO 2 PSAP
PSAP
112 eCall Trigger (eCall sensors or manual)
Transmission over vehicle bus
In-vehicle software triggers 112 call
in-vehicle communication module initiates 112 call and send MSD
Receive 112 call and MSD
Enrich 112 call with CLI, celluar location and MSD
Forward 112 voice, CLI, celluar location and MSD to PSAP
Answer 112 voice call, decode and visualise celluar location and MSD

Figure 44 The six domains of an eCall service chain

been complemented with other important requirements, objectives and design constraints (R-O-Cs) for the selection of an eCall modem candidate for standardization in 3GPP.

1. ***eCall MSD Data Transmission requirement*** Emergency calls may be supplemented with emergency related data. This data includes the accurate geographic location of automatically or manually activated emergency calling device. The 3GPP has developed a set of requirement for in-band modem solution for eCall.

   (a) The data may be sent prior to, in parallel with, or at the start of the voice component of an emergency call.

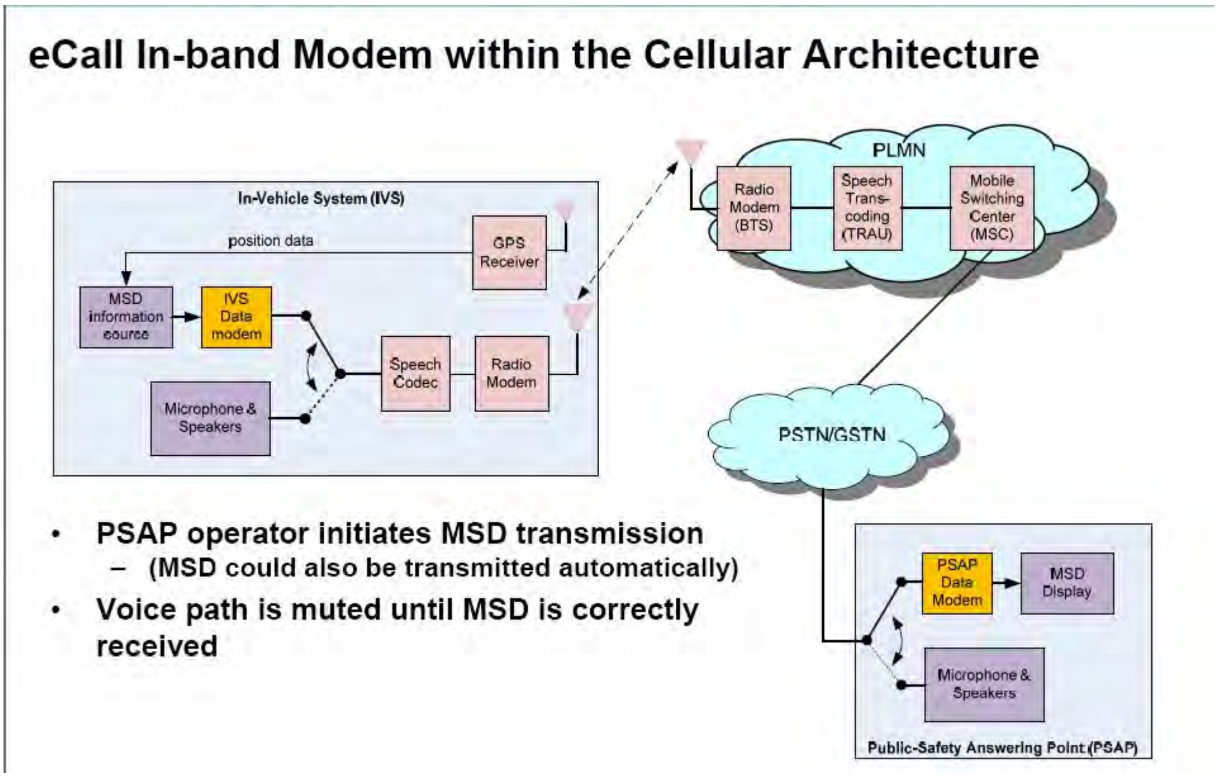
   (b) During the established emergency call, the protocol transmit eCall data in whole or parts at any time during the voice call, to the same PSAP or a second or further one.

   (c) The realization of the transfer of data during an emergency call shall minimize changes to the originating and transmit network.

   (d) Both the voice and data component of the emergency call shall be routed to the same PSAP or designated emergency call center.

   (e) The transmission of the data shall be acknowledged and if necessary data shall be retransmitted.

2. ***eCall Minimum Set of Data (MSD)*** The Minimum Set of Data (MSD) sent by the in vehicle system shall not exceed 140 bytes as shown in Figure 46. The eCall recommends that the performance criteria related to the timing in the eCall service chain be kept shorter. The MSD should typically be made available to the PSAP within 4 seconds measured from the time when end to end connection with the PSAP is established. However, the minimum transmission time for full MSD


eCall In-band Modem within the Cellular Architecture

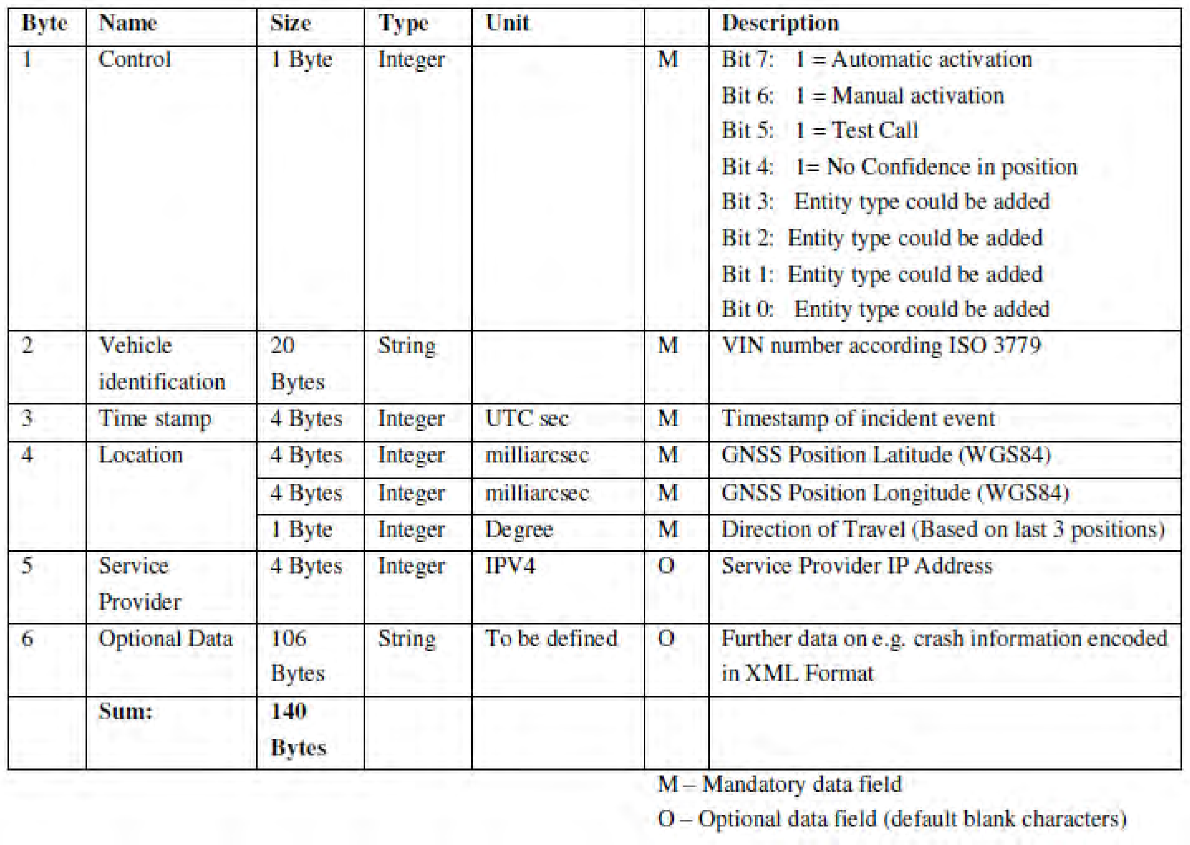
- PSAP operator initiates MSD transmission
  - (MSD could also be transmitted automatically)
- Voice path is muted until MSD is correctly received

Figure 45 In-Band Modem Solution

depends on the amount of data in the MSD. The contents of the MSD are shown in Figure 46:

3. ***eCall via CTM Protocol*** The existing 3GPP standard for CTM as specified in 3GPP TS 26.226, TS 26.230 and TS 26.231, developed to fulfill the lawful requirements for text telephony in emergency calls as set up by the United States of America, allows the robust transmission of textual characters via de facto any voice channel. The following description assumes that the eCall data are structured in indexed blocks of a priori known version (v), index (i) and size and an automatically generated In-Vehicle System (IVS) Content list marks all blocks available within a specific IVS. If this is not the case – the decision is within the responsibility of CEN – then the protocol here may divide the full, unstructured MSD of 140 bytes in equal sized smaller blocks, e.g. of 28 byte length each. The block size of 28 bytes is a compromise between transmission speed and radio robustness. Bigger blocks reduce the overhead, thus reduce the transmission time, but increase the error probability under severe radio conditions.

4. ***eCall transmission*** The proposal here is based on the assumption that the existing CTM implementations in terminals are used without any modification. In order to avoid these CTM intrinsic effects, the eCall data are first re-coded by the eCall sender by using the following scheme: (7 bytes x 8 bit) are coded as (8 bytes x 7 bit) that can be transmitted by CTM without problems. Then a constant offset of 0x20 is added to every intermediate byte in order to assure that only values in the



Byte	Name	Size	Type	Unit		Description
1	Control	1 Byte	Integer		M	Bit 7: 1 = Automatic activation Bit 6: 1 = Manual activation Bit 5: 1 = Test Call Bit 4: 1= No Confidence in position Bit 3: Entity type could be added Bit 2: Entity type could be added Bit 1: Entity type could be added Bit 0: Entity type could be added
2	Vehicle identification	20 Bytes	String		M	VIN number according ISO 3779
3	Time stamp	4 Bytes	Integer	UTC sec	M	Timestamp of incident event
4	Location	4 Bytes	Integer	milliarcsec	M	GNSS Position Latitude (WGS84)
		4 Bytes	Integer	milliarcsec	M	GNSS Position Longitude (WGS84)
		1 Byte	Integer	Degree	M	Direction of Travel (Based on last 3 positions)
5	Service Provider	4 Bytes	Integer	IPV4	O	Service Provider IP Address
6	Optional Data	106 Bytes	String	To be defined	O	Further data on e.g. crash information encoded in XML Format
	Sum:	**140 Bytes**				

M – Mandatory data field
O – Optional data field (default blank characters)

Figure 46 eCall Minimum Set of Data (MSD) range of [0x20,0x9F] are finally obtained. In this way no CTM-reserved characters and no values in the range of [0xA0, 0xFF] occur in the recoded data. This 7-to-8 recording leads to a marginal and constant increase in transmission time by factor 1,143. For the same reason no eCall-Header byte, eCall-Index byte or eCall-CRC byte may take a value outside the range of [0x20, 0x9F]. The eCall receiver reverts all this re-coding to gain the original eCall data back.

Two different types of eCall packets are:

- ***PSAP-Pull protocol*** The transmission time for an eCall PSAP-Request is about 1.5s.

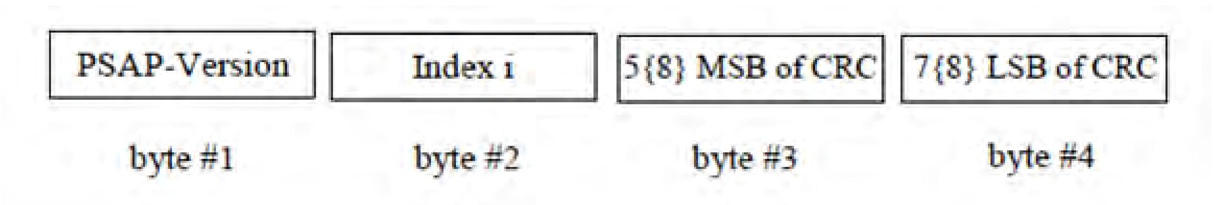


Figure 47 eCall PSAP packet

  (a) Byte 1 contains the PSAP-Request-header Values between 0x20 to 0x5F (64 values) (128?) are reserved. The PSAP-Request Header identifies the packet as PSAP-originated and indicates in addition the eCall Versions the PSAP is able to operate on.
  (b) Byte 2 contains the index i of the requested eCall-data block. Up to 128 {256} different data block indices can be coded as 0x20+ [0 127] {[0 255]}.
  (c) Byte 3 and byte 4 contain a 12 bit or 16-bit CRC

- ***PSAP-Push protocol*** The transmission time for an eCall IVS-data block of k bytes is approximately: T (k) =1400ms + (100*k) ms.

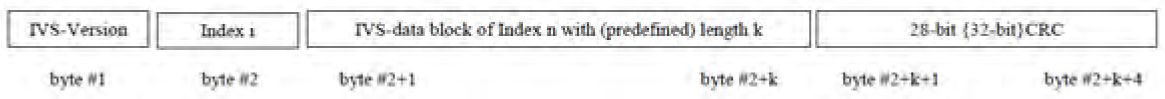


Figure 48 eCall IVS packet

  (a) Byte 1 contains the IVS-Data Header Values between 0x60 to 0x9F (64 values) (128?) are reserved. The IVS-Data Header identifies the packet as IVS-originated and indicates in addition the common eCall Versions the IVS is able to operate on,
  (b) Byte 2 contains the index i of the transmitted eCall data block. Up to 128 {256} different data block indices can be coded as 0x20+[0 127]{[0 255]}
  (c) Bytes 2+1 to 2+k contain the k bytes of the 7-to-8 recoded eCall data block (i). Bytes 2+1 to 2+k contain the k bytes of the original eCall data block (i).
  (d) Bytes 2+k+1 to 2+k+4 contain the 28-bit {32-bit} CRC in 4 parts of 7 (8) bit each.

A.2.2 Conclusion

As concluded by European Commission, an in-band modem is the only appropriate solution to provide a secure solution for the transmission of emergency data that maximizes the service coverage area, offers a prioritized and fast transmission of the MSD, and minimizes the required modifications to the cellular core networks and the public switched telephone network (PSTN). However, in order to protect the MSD to be forged, the in-car communication needs to be secured. This is done be EVITA security architecture framework, comprising a detailed description of software and hardware security modules. This provides a comprehensive security framework for the flexible and optimized deployment of security functionality within automotive on-board networks.

A.3 eTolling

A.3.1 Identification friend or foe (IFF)

In order to understand he concepts of electronic toll collection it is helpful to take a look on its basis. One basis of ETC is a military system called identification of friend or foe (IFF) that was originated during World War 2 with the sole purpose to enable U.S. secondary radars to identify U.S. aircraft from enemy aircraft by assigning a unique identifier code to U.S. aircraft transponders. So the initial idea was to distinguish between enemy and friend [62]. Five major modes of operation are currently in use by military aircrafts plus one submode:

1. C Mode 1 is a nonsecure low cost method used by ships to track aircraft and other ships.
2. C Mode 2 is used by aircraft to make carrier controlled approaches to ships during inclement weather.
3. C Mode 3 is the standard system also used by commercial aircraft to relay their position to ground controllers throughout the world for air traffic control (ATC).
4. C Mode 4 is secure encrypted IFF
5. C Mode 5, Levels 1 and 2 are crypto-secure with enhanced encryption, Spread Spectrum Modulation, and Time of Day Authentication.
   (a) Mode 5, Level 1 is similar to Mode 4 information but enhanced with an Aircraft Unique PIN number.
   (b) Mode 5, Level 2 is the same as Mode 5 level one but includes additional information such as Aircraft Position and Other Attributes.
6. C Mode "C" is the altitude encoder.

Technically speaking Cross-band beacon is used, interrogation pulses are at one frequency and the reply pulses are at a different frequency. The most popular frequency pair used in the U.S. is 1030 MHz and 1090 MHz.

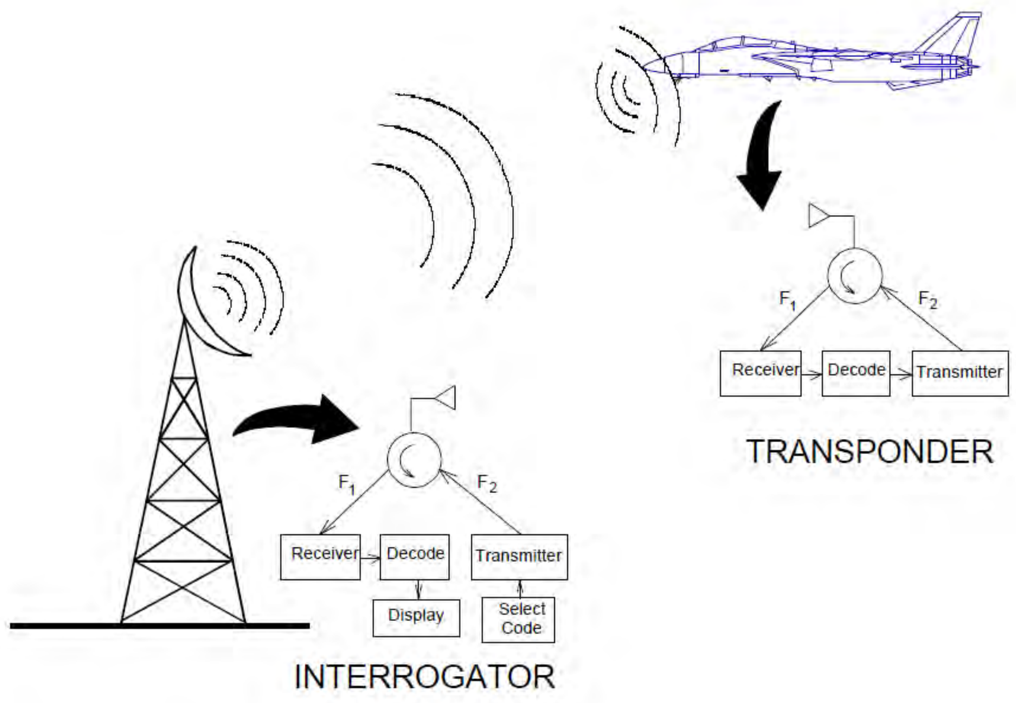
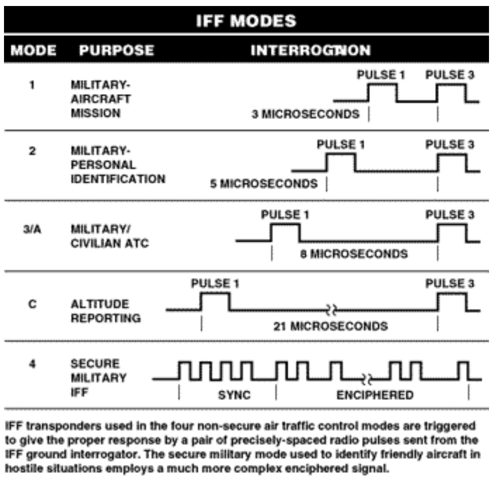


Figure 49 Identification friend or foe

The secondary radar transmits a series of selectable coded pulses. After that the aircraft transponder receives and decodes the interrogation pulses. If the interrogation code is correct, the aircraft transponder transmits a different series of coded pulses as a reply, e.g. the aircraft was told to squawk a four digit number such as "4732". The altitude encoded transponder provides the aircraft altitude readout to the ground controllers' display along with the coded response identifying that particular aircraft. Because the main purpose for IFF is to distinguish enemies from friends, it is essential that hostile forces are not able to use the system to identify them as friendly even if the physical IFF equipment should fall into their hands.

Therefore the secure mode is established. This mode uses a very long challenge word which contains a preamble that tells the transponder it is about to receive a secure message. The challenge itself is encrypted at the interrogator by a separate device that uses various mathematical algorithms to put it in a secure form. The transponder routes the ensuing challenge to a separate device that uses the inverse algorithms to decode the challenge. In effect, each challenge is telling the transponder to respond in a certain way. If the transponder cannot decipher the challenge, it will not be able to respond in the proper way and thus will not be identified as a friend. To prevent unauthorized use of either the interrogation equipment or the transponders if they should fall into hostile hands, a key code must be periodically entered into each device. To eliminate the chance of a random guess by a hostile target corresponding with the proper response, each identification consists of a rapid series of challenges each requiring a different response that must be



IFF transponders used in the four non-secure air traffic control modes are triggered to give the proper response by a pair of precisely-spaced radio pulses sent from the IFF ground interrogator. The secure military mode used to identify friendly aircraft in hostile situations employs a much more complex enciphered signal.

Figure 50 IFF modes

correct before the target is confirmed as a friend. A very high degree of security to the identification system is ensured through the use of key codes and powerful cryptographic techniques [8]. Due to the fact that it is military equipment that we are talking about, all further cryptographic aspects can be viewed as secret and are not publicly available.

A.3.2 Electronic toll collection (ETC)

As we stated earlier ETC is an adaptation of the IFF concept. Nowadays several countries around the world and U.S. states have implemented ETC on their routes enabling the collection of highway tolls without the existence of manually operated toll booths and with no more marginal speed reduction of the passing by vehicle. Within most technologies vehicle identification is fulfilled with radio-frequency identification, where an antenna at the toll gate communicates with a transponder on the vehicle via Dedicated Short Range Communications (DSRC). RFID tags have proved to have excellent accuracy, and can be read at highway speeds. The following listing provides an overview about existing ETC systems in Europe:

Western Europe

- Austria – ASFINAG
- Austria – go-maut for the national Autobahn network
- Germany – LKW-MAUT for trucks on Autobahns (TOLL COLLECT)
- Italy – TELEPASS on Autostrade motorways
- France – Telepeage usually branded liber-t on motorways (run by the Federation of French Motorway Companies) (ASFA)
- Spain – VIA-T or Telepeaje
- Portugal – Via Verde (all tolls)
- Switzerland (LSVA)

United Kingdom and Ireland

- Ireland – eToll national standard
- Ireland – Eazy Pass National Toll Roads implementation of eToll
- United Kingdom – Dart-Tag for the Dartford Crossing
- United Kingdom – London congestion charge in London
- United Kingdom – Fast tag Mersey tunnels: Queensway Tunnel and Kingsway Tunnel
- United Kingdom – M6 Toll tag in the Midlands
- United Kingdom – Severn TAG for the Severn Bridge crossing and Second Severn Crossing
- United Kingdom – Tamar Bridge
- United Kingdom – Permit for the Tyne Tunnel linking North and South Tyneside

Scandinavia

- Norway – AutoPASS in most of the country
- BroBizz for the Oresund and Great Belt bridges in Denmark/Sweden
- Sweden – Stockholm congestion tax in Stockholm

Eastern Europe

- Croatia – ENC, on all tolled highways in Croatia (autocesta)
- Turkey – OGS
- Slovenia – ABC
- Czech Republic – premid for trucks on highways

What we can derive from this enlisting above is the fact that talking about ETC means talking about a variety of systems each of them partially or completely incompatible even within a country. Especially in Europe we find a lot of different solutions. Therefore, the EU started harmonization work early with the directives 1999/62/EC and 2006/38/EC on the charging of heavy goods vehicles for the use of certain infrastructures which determines Common rules on how EU states may charge heavy goods. Since May 2004 the directive 2004/52/EC is in force that defines the conditions for interoperability of electronic road toll systems in the EU. One outcome of the directive is that all new electronic toll systems brought into service after 1 January 2007 must, for carrying out electronic toll transactions, use one or more of the following technologies:

- satellite positioning
- mobile communications using the Global System for Mobile Communications (GSM) General Packet Radio Service (GPRS) standard (reference GSM TS 03.60/23.060)

- 5,8 GHz microwave technology

As a conclusion in order to guarantee the interoperability of European ETC a European Electronic Tolling Service (EETS) was created. The main slogan of EETS is “one single contract – one single on board unit” which represents the basic principle. In 2005 the Road Charging Interoperability (RCI) program (6th FP) led by ERTICO was started. The RCI project developed an open and integrated framework enabling road charging interoperability at a technical level, based on the key existing and planned road charging deployments in Europe (AUTOPASS, ASFINAG, LSVA, TELEPASS, TIS, TOLL COLLECT, VIA-T and VIA VERDE). The framework was implemented and tested in field trials at six sites, namely Austria, France, Germany, Italy, Spain and Switzerland (LSVA). The RCI specification defines the following technical entities and roles:

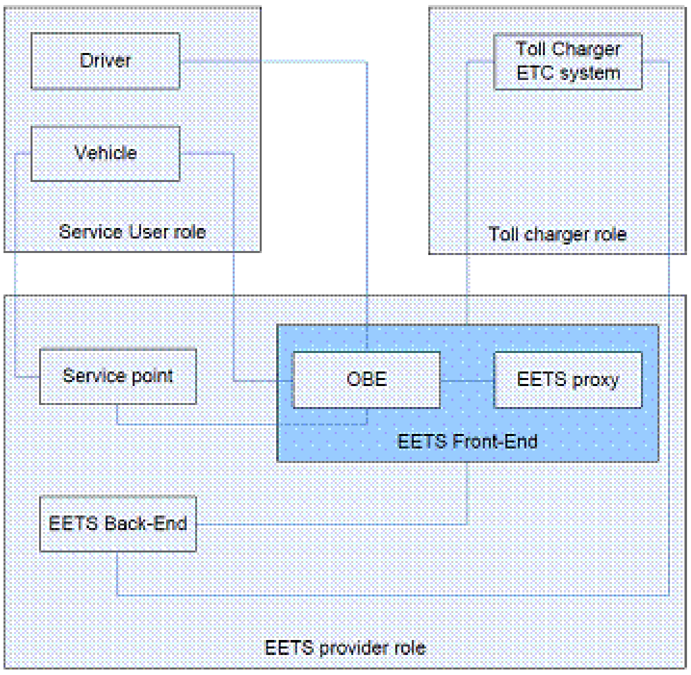


Figure 51 RCI technical entities and roles

Note that the EETS Front-End is a term describing the fact that there may be two architectural variants namely an OBE that:

1. 1. Sends most of its sensor data to a proxy as part of the central equipment of the EETS provider that processes this data in order to detect the charge objects and to prepare the charging data

2. 2. Processes itself most of the sensor data, detecting the charge objects and events and storing charging data records in its memory.

In the latter case the EETS Front-End simply is a single OBE fulfilling the whole role itself without a proxy. The following figure shows the overall system overview of functional elements and reference points ensuring interoperability that are needed for European interoperable road charging. Those reference points (interfaces) that need to be standardized in order to guarantee interoperability among different implemented solutions are marked as red arrows. The other interfaces do not need – although critical to be part of the solution – to be implemented according to an open standard, but could in principle be implemented on the basis of proprietary specifications by suppliers without undermining the interoperability concept.

All functions that handle vehicle related information are bundled in the *V. VehicleProvidedData block*. Examples are: tacho signal, vehicle details (if communicated via CAN bus) like e.g. "trailer attached". This lets us see that a connection of the OBE to the CAN Bus of the vehicle is envisioned. Nonetheless the interface is not target of standardization but may be implemented in a proprietary way by each supplier. The reference point is defined as follows:

Within the RCI project two supplier consortia provided two prototypes. Each prototype was installed in a truck for field test in order to verify the feasibility of the implementation of the RCI specification. In order to understand the interfaces that were used to connect the OBE to the vehicle we will take a brief look at one implemented prototype provided by the FELA, Elem Q-Free (F.E.Q for short) supplier consortia. Each consortia partner covered distinct roles:

- FELA taking the main responsibility for software development and most hardware development components mainly the main unit with its GPS tolling application
- ELEM taking the main responsibility for certain hardware developments, housings, production of prototypes and FAT
- Q-Free taking the main responsibility for all CEN DSRC issues and delivering DSRC modules for integration

Important Partners for the development were:

- Autostrade delivering UNI DSRC modules integrated to work in the Telepass context
- Toll Collect delivering IR test beacons to allow for testing the IR transaction

The main unit of the F.E.Q. OBE was implemented according to the scheme in Figure 54.

The OBE itself contains the following connections:

- 10Pin Connector to connect to the vehicle interface,
- 10Pin Connector to connect to the DSRC communication module,
- FAKRA type connection for the external GPS antenna,
- FAKRA type connection for the external GSM antenna,
- Ethernet connection (for development only).

Concerning the vehicle interface the minimum interface regarding distance input is a connection towards a tachograph or a vehicles speed pulse. This was chosen as older vehicles do not have a standard CAN interface available that could be used instead of the tacho or speed pulse. Nevertheless CAN is available on the RCI prototype OBE. The following connections are available

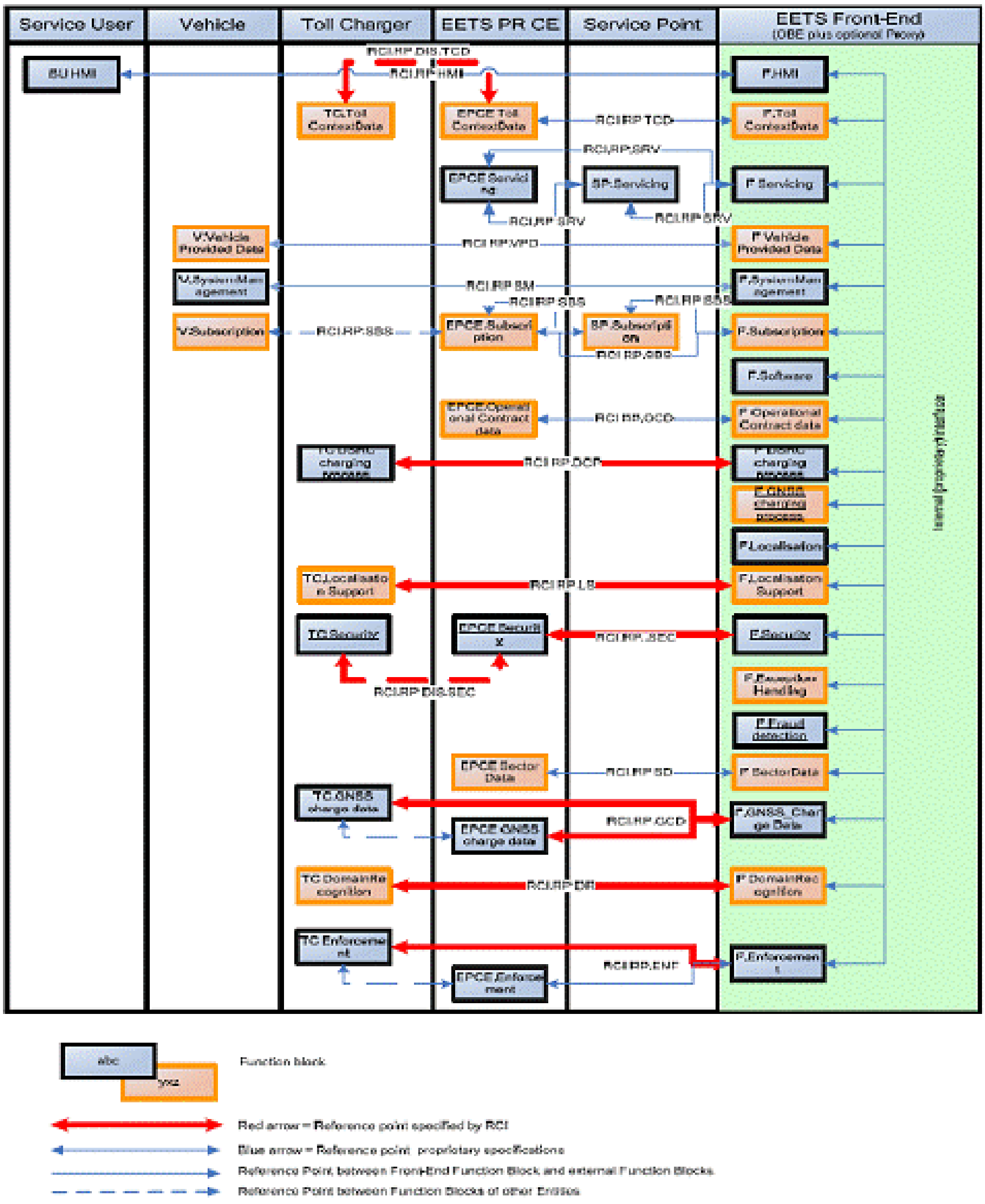


Figure 52 Interfaces between technical entities

- P1 = Constant Power feed 24Vdc, Red wire (needed)
- P2 = Illumination On Terminal, Yellow (not needed by F.E.Q.)
- P3 = Ignition Terminal, Green wire (needed)

Interface	RCI.RP.VPD (Vehicle Provided Data)
Description	This reference point deals with physical as well as protocol issues. E.G the minimum required for installation into older vehicles is physical signals from tacho or similar signal like speed pulse. On newer vehicles there may often be a bus system available like for instance a CAN-Bus interface
Involved Entities	Vehicle, OBE
Involved Function-Blocks	RCI.FB.F.VehicleProvidedData, RCI.FB.V.VehicleProvidedData
Message Set	Minimal OBE physical connection support: • Tacho pulse or other equivalent pulse signal • Ignition key signal, Trailer sensor, Clock CAN Bus support: • GetVehicleDistance • GetVehicleTrailerStatus
Protocol	For CAN-Bus the following is suggested: • FMS Standard Interface for Trucks • ODB may be used for Personal Cars
Communication Technology	Physical connection, CAN-Bus
To be specified	For new Vehicles there should be an easy possibility for installers to connect the minimum required signals in an easy way. Vehicle manufactures can provide these signals in there proprietary way, but should provide a minimum standardized set of signal connections, that allow for installation of tolling OBE's: FMS is supported by: Daimler Chrysler, MAN, Scania, Volvo, DAF, Iveco
Existing Specifications Reference	LSVA Specification, TollCollect Specification

Figure 53 Reference point RCI.RP.VPD

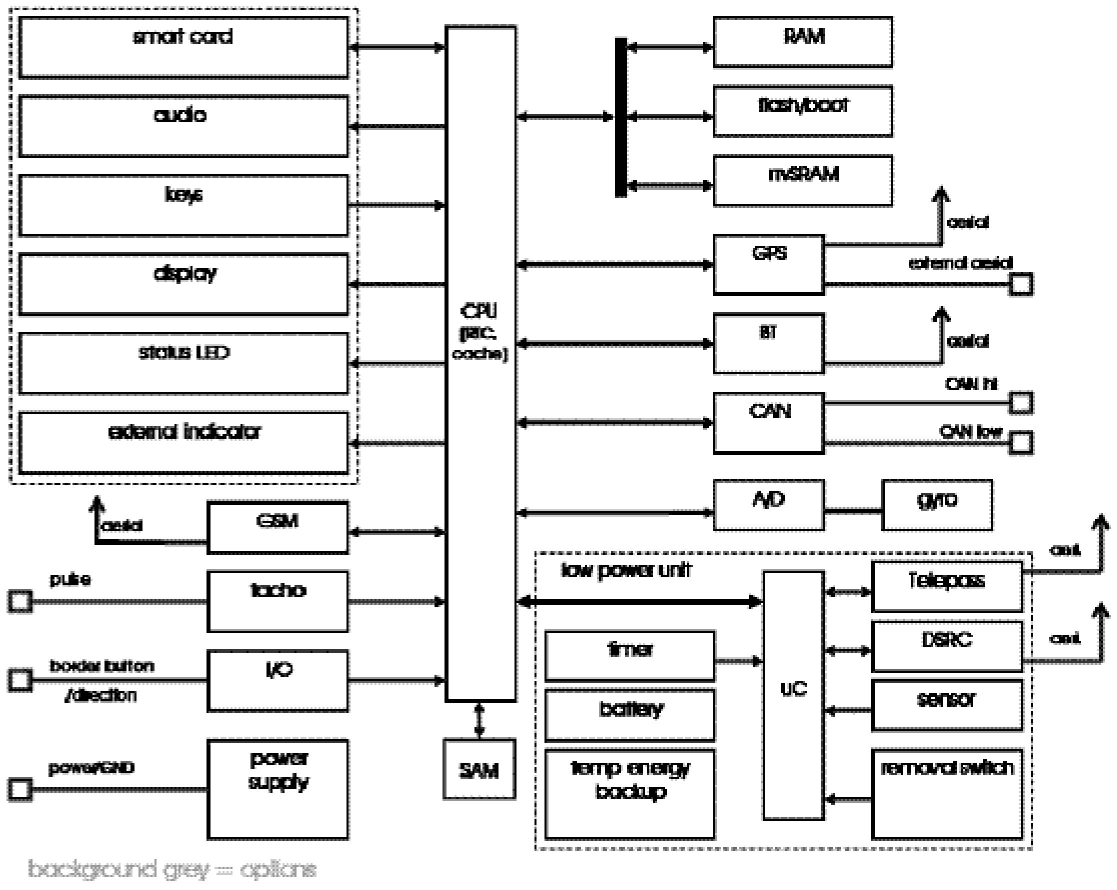


Figure 54 Implementation scheme

- P4 = *no wire*
- P5 = Earth from battery for tachograph, Black wire (needed)
- P6 = Earth from battery for Illumination, Brown (needed)
- P7 = Tachograph speed source signal for OBU, Blue (needed)
- P8 = CAN+ (not available on all Vehicles, not needed by F.E.Q)
- P9 = CAN- (not available on all Vehicles, not needed by F.E.Q)
- P10 = CAN Ground (not available on all Vehicles, not needed)
- P11 = backward driving signal (optional, recommended)
- P12 = Trailer sensor signal (optional, recommended)

The second prototype that was implemented (T2ASK) conforms to the following overall structure of external and internal interfaces:

This results in the following vehicle interface (EI1, EI2):

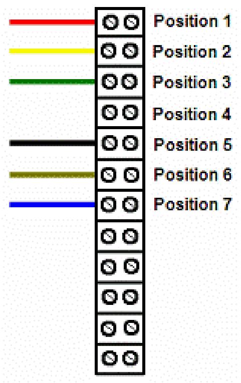


Figure 55 Prototype 1 connections

- P1 = Constant Power feed 12 or 24Vdc, Red
- P2 = not for RCI
- P3 = Ignition Terminal, Green
- P4 = *no wire*
- P5 = Earth from battery, Black
- P6 = Not for RCI
- P7 = Odometer distance/speed signal for OBU, Blue
- P8 = CAN+ (maybe not available on certain Vehicles)
- P9 = CAN- (maybe not available on certain Vehicles)
- P10 = CAN Ground (maybe not available on certain Vehicles)
- P11 = *no wire*
- P12 = Trailer sensor signal

Note that in both cases the connection to a CAN-bus was designed but not used as only the minimum requirements were implemented.

A.3.3 Conclusion

As for now eToll applications are rarely integrated into the main architecture of a vehicle. The devices are operated in an nearly isolated manner and are mostly sealed physically by the manufacturer. Security aspects have to be implemented by the manufacturers, as a secure platform is not available in current scenarios. The EVITA project fills this gap by providing a hardware security module that guarantees high platform security an opportunity for secured key storage, the secure integration of local devices and secure communication with the infrastructure (V2I) or with mobile communication networks.

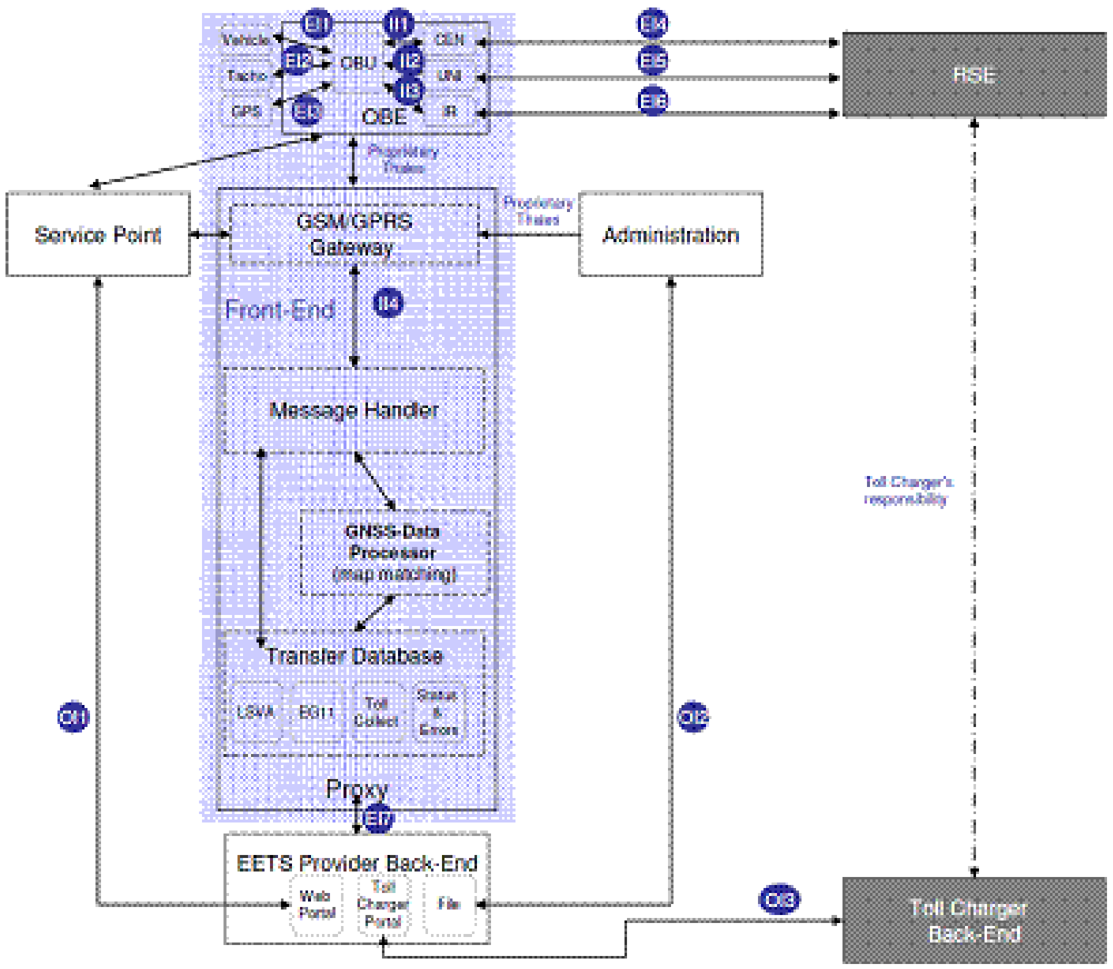


Figure 56 Prototype 2 interfaces

Furthermore the multi-purpose EVITA module can help manufacturers to reduce production cost of their equipment as they can rely on the security features of the EVITA module. Mechanisms that are already available within the EVITA module can be reused by eToll devices that are equipped with an interface to the in-car system.

A.4 Common Transport Protocol of the EASIS project

Different gaps have been identified after comparing the requirements of data exchange between different domains in in-car network and the mechanisms provided by existing standards as CAN, FlexRay. Those gaps are summarized as: the shortage on reliable transport protocol for inter-domain communication, the missing link to vehicle-external entities, the missing security mechanisms and the missing unique addressing scheme. In order to fill these gaps, the Common Transport Protocol CTP of the EASIS project [43] specifies a transport protocol independent of typical underlying transport protocols as CAN, FlexRay or Transmission Control Protocol (TCP)/IP. Following features represent the benefit from CTP:

1. Network Layer
   - Independence of the underlying network
   - Abstraction of the network topology
   - (Inter-)network wide addressing scheme
   - Addressing space of 15 bit
2. Transport Layer
   - Multiplex transport connections: In order to save cost and performance when many connections are needed, EASIS recommends to multiplex the transport connections on to the same network connection. The transport layer should make the multiplexing transparent to the session layer. However, in [43] there is no explanation how the multiplexing is achieved.
   - Provides reliable datagram service
   - Flow control

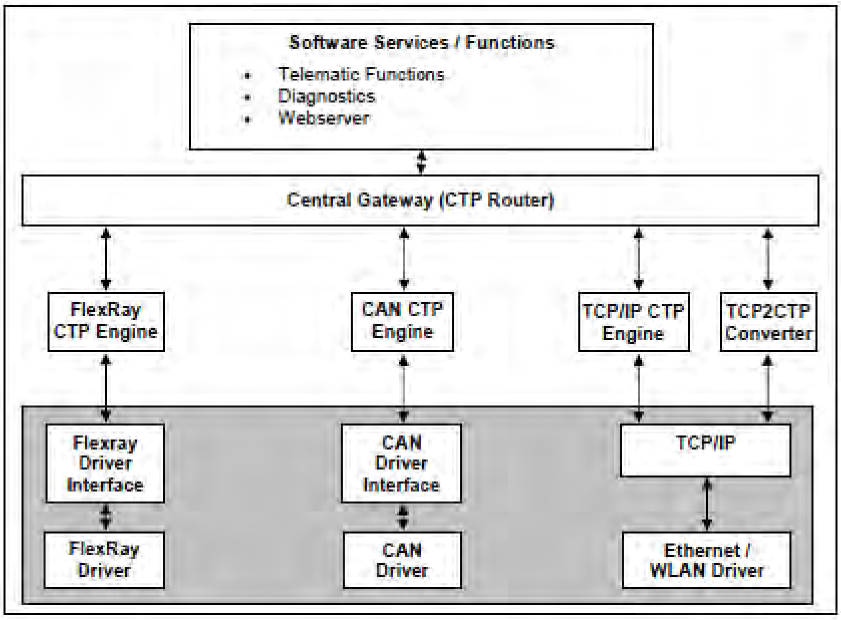


Figure 57 Central gateway with CTP router

Figure 57 provides a view of a central gateway architecture based on CTP. This architecture enables the connection of different domains of the in-car network. Therefore, applications on ECUs can communicate through this gateway independently of their network domain. The gateway architecture is based on the Automotive Open System Architecture (AUTOSAR) architecture which is foreseen inter-domain communication. Furthermore, the AUTOSAR architecture provides support for the different in-car bus systems. The AUTOSAR architecture description can be found in [9].

Figure 58 from [43] shows the EASIS gateway architecture as extension of AUTOSAR. In the following paragraphs, the specification of the CTP from [43] is described.

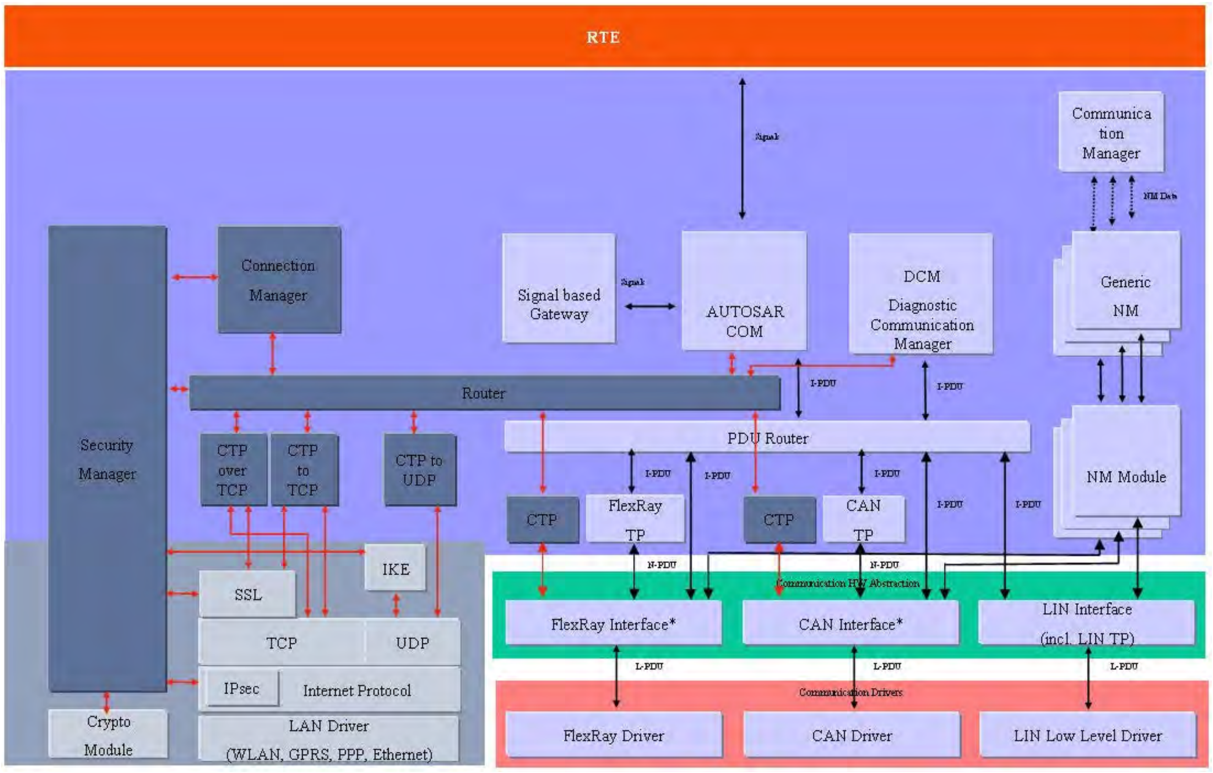


Figure 58 EASIS Architecture as Extension of the AUTOSAR Architecture

Message size To support communication through the FlexRay Transport layer, CTP can transmit message with a size up to 4 GBytes.

Address space CTP uses an address space of 15 bits for source and target address.

CTP frame types CTP frames provide a target and source address as well as the network layer protocol control information N-PCI as header information. There are 12 types of CTP frames. Depending on the CTP frames, different types of messages are allowed. The format of the CTP frame in the network depends on the underlying bus system.

Flow control, acknowledgement and error indication CTP supports flow control to send multi-frame messages as CAN and FlexRay does. It supports the acknowledgement of messages. It also supports datagram services which do not require an ACK. The differentiation between both is done by using different frame types.

CTP over CAN CTP provides 15 bits address space by using an additional byte of the payload of the CAN frame. But the fact that CAN uses the CAN ID for bus arbitration makes it impossible to guarantee that each competing identifier is unique during bus arbitration. In order to differentiate between CTP and CAN, the extension of the CAN identifier provides by the ISO 15765-3 norm is applied. Therefore CTP uses the complete address extension for the source address.

CTP over FlexRay CTP uses the payload of the FlexRay message for the header information. CTP uses the payload flag provided by FlexRay to indicate an uneven number of bytes in the payload to be consistent with the fact that FlexRay only support the transmission of frames with an even number of bytes. Therefore if the payload flag is set, the size of the CTP Payload is set to FlexRay payload minus one.

CTP over TCP/IP In order to convert CTP to TCP, the payload and the header information from CTP have to be converted to a TCP packet data unit. The challenge is to extract the CTP address information and convert it to TCP address information. For the mapping the IP address as well as the TCP/User Datagram Protocol (UDP) port are provided by TCP. Since the use of IP addressing complicates the integration of the in-vehicle CTP network into a global network, CTP uses the port address (size of 16 bits). Specifically the upper part of the port address is used to map the CTP addresses, since the lower part of the port address space is reserved for certain standard services. The advantage of this approach is the use of one single IP address to integrate the in-vehicle network in a global network. To send messages from a CTP entity to a TCP entity or vice versa, a gateway is used for the conversion of the protocols:

- Information flow from TCP to CTP: The receiver is a CTP entity but the Gateway is a TCP-to-CTP converter which means a TCP entity. This has the inconvenience of complicating an end-to-end communication between TCP sender and CTP receiver. In fact, the TCP receiver (Gateway) sends an acknowledgment to the sender as soon as the connection is established, which leads to confusion, since it does not know then if he is communicating with the target CTP entity. To resolve the issue the TCP stack has to be adapted to first send the acknowledgment when the CTP receiver sent it to the gateway.

- Information flow from CTP to TCP: The receiver is a TCP node but the Gateway is a CTP-to-TCP converter, which means a CTP entity. In this case, the CTP gateway will first send an acknowledgment to the CTP sender after he received one acknowledgment from the target TCP entity.

Tunneling To enable two CTP end points to communicate with each other through a network using TCP, CTP proposes a tunnel as shown in Figure 59 from [43]. Since in TCP the sending application does not have a direct influence on the number and size of TCP packets, the receiver does not know which data has been transmitted by which TCP packet. To be able to identify the CTP headers, the CTP headers needs to know where a CTP packet ends and when the next starts. Therefore in order to realize the tunnel, additional information on the length of the packet segments has to be added to the TCP stream. Furthermore the limitation of CTP addresses is an issue if CTP is used to communicate with remote peers. [43] proposed the use of alias Addresses, which requires a network address translation.

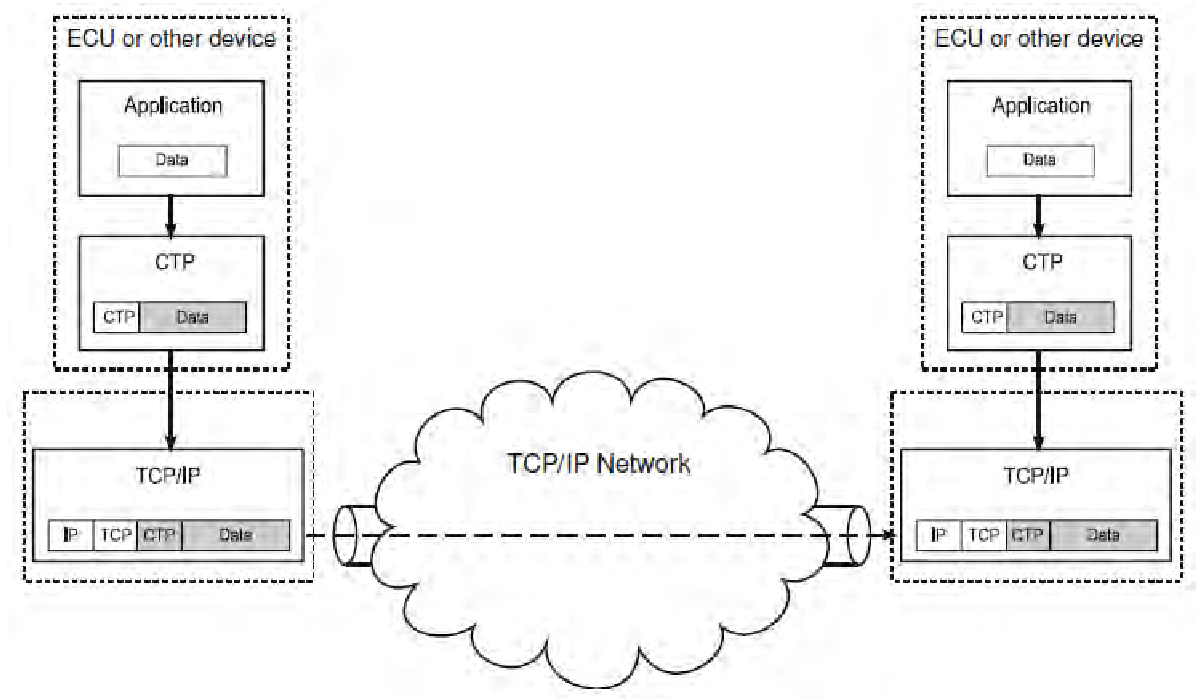


Figure 59 Tunneling of CTP over TCP

A.5 Flashing and Diagnosis

A.5.1 HIS specification

The name HIS stands for Hersteller-Initiative Software. It is a consortium of German car manufacturers (BMW, AUDI, Daimler, Porsche and Volkswagen) with the goal to achieve and use joint software standards in vehicles. The working group *flash programming* of HIS specifies in [46] a flash programming process based on [26]:

- Optimization of the flash process
- Standardization of relevant ECU software modules
- Software Security

In Figure 60 you can see the HIS software architecture for the flash loader process. In [46] the flash loader process for ECUs is described. The process includes a security module which is specified in [45]. Security features include standardized routines, authorization of the programming device to the ECU and testing of authenticity and integrity of the firmware to be flashed. In [45] two digital signatures for this purpose are recommend.

- The first signature of the flash container is generated by the supplier and shipped with the flash product. This is verified by the manufacturer and proves that the software was indeed released from the supplier.
- The manufacturer generates a second signature of the flash container. This is verified and documented in the control unit, and proves that the software was released by the manufacturer.

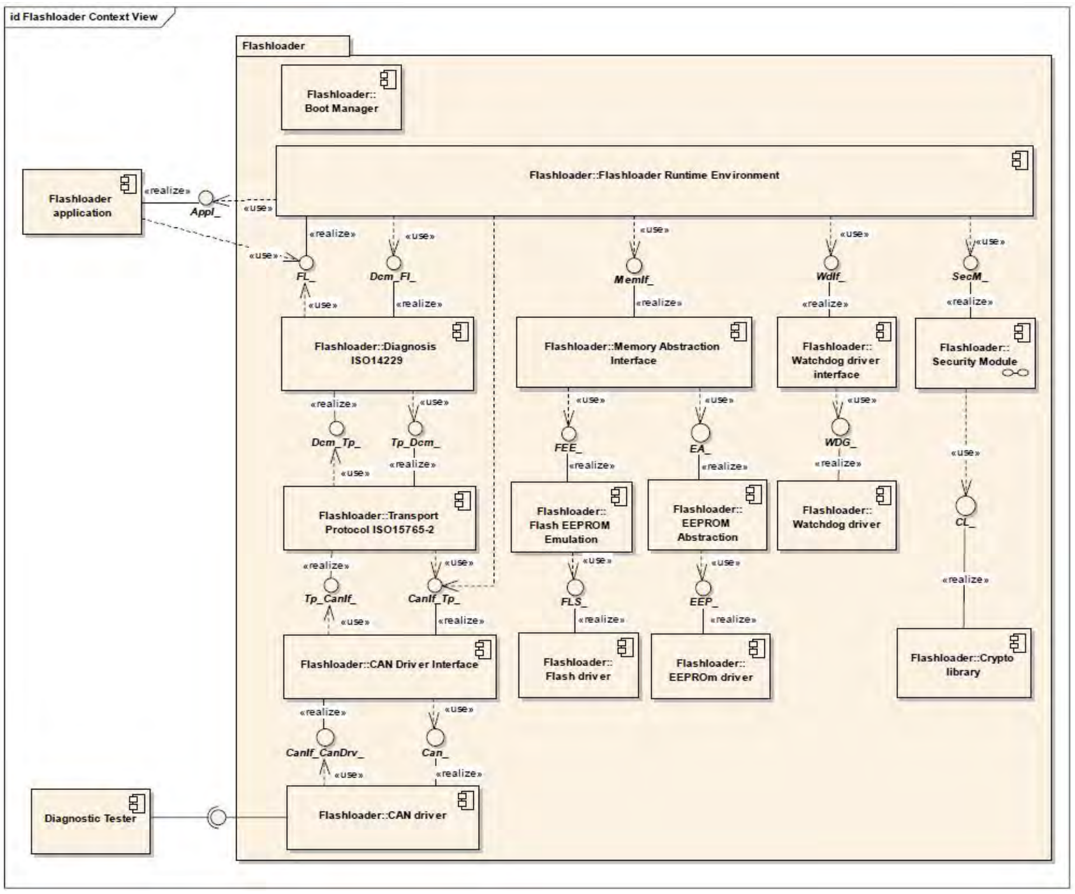
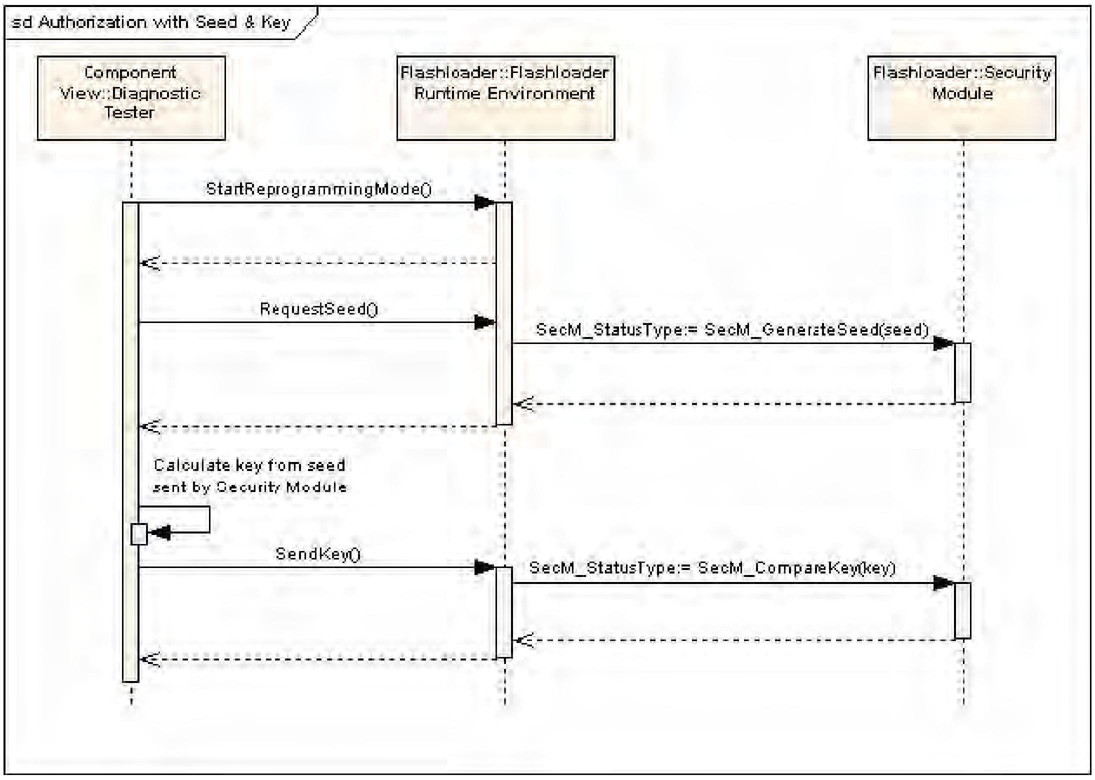

id Flashloader Context View
Flashloader
Flashloader:: Boot Manager
Flashloader::Flashloader Runtime Environment
Flashloader application
«realize»
Appl_
«use»
FL_
Dcm_Fl_
MemIf_
WdIf_
SecM_
Flashloader::Diagnosis ISO14229
Flashloader::Memory Abstraction Interface
Flashloader:: Watchdog driver interface
Flashloader:: Security Module
Dcm_Tp_
Tp_Dcm_
FEE_
EA_
WDG_
CL_
Flashloader::Transport Protocol ISO15765-2
Flashloader:: Flash EEPROM Emulation
Flashloader:: EEPROM Abstraction
Flashloader:: Watchdog driver
Tp_CanIf_
CanIf_Tp_
FLS_
EEP_
Flashloader::CAN Driver Interface
Flashloader:: Flash driver
Flashloader:: EEPROm driver
Flashloader::Crypto library
CanIf_CanDrv_
Can_
Diagnostic Tester
Flashloader::CAN driver

Figure 60 HIS Flashloader Software Architecture

Significant steps in the flashing process which involve security are: The authorization step, the download procedure step and the verification step, which are explained in the following paragraphs.

HIS: Authorization The HIS solution for authorization is demonstrated in Figure 61. The Security Module must verify the authenticity of the diagnostic tester with a seed key algorithm before any download activities start. To start the authorization, the diagnostic tester requests an integer value as a seed from the ECU. The FLaRE (FLashloader Runtime Environment) passes the request to the Security Module. FLaRE is part of the HIS standard Flashloader. The Security Module returns the seed and calculates the corresponding key value. The external programming tool calculates the key, too, and passes it through the FLaRE to the Security Module. If the two calculated values match, the ECU is *unlocked* and the download procedure can be started. Otherwise the ECU remains in the state *locked* and the FLaRE ensures that flashware download cannot be started.

HIS: Download procedure Figure 62 shows HIS proposed download procedure of firmware. The encrypted firmware is downloaded block by block (logical block). Each of those blocks is divided into segments, which are a set of bytes containing a start address and a length. The start address and the length of each segment is sent to the security



sd Authorization with Seed & Key
Component View::Diagnostic Tester
Flashloader::Flashloader Runtime Environment
Flashloader::Security Module
StartReprogrammingMode()
RequestSeed()
SecM_StatusType:= SecM_GenerateSeed(seed)
Calculate key from seed sent by Security Module
SendKey()
SecM_StatusType:= SecM_CompareKey(key)

Figure 61 HIS session handling

module during the segment initialization. For one block, a download request is sent from the diagnostic tester to the security module. The security module initializes the decryption service and sent an answer to the diagnostic tester. The download then starts segment by segment. In [45] it is not described whether the decryption is done segment by segment. Only the following is stated *decryption is currently not implemented.*

HIS: Verification After the download procedure HIS recommends in [45] a verification with 3 optional algorithms. Figure 63 shows the verification procedure component wise.

- Use of CRC32: The logical block has a Cyclic Redundancy Code (CRC) written into flash. At the end of the download the CRC is computed and compared.
- Use of MAC [61]: The authenticity is verified by comparing the transmitted MAC with the calculated value.
- Use of Signature [38]: The authenticity/integrity is checked by verifying the RSA signature.

Figure 64 resumes all the possible steps for verification.
The flashing process provides by HIS provides a good basis for the OEMs, but the recommended algorithms have to be updated. Furthermore an important security goal as freshness is missing.

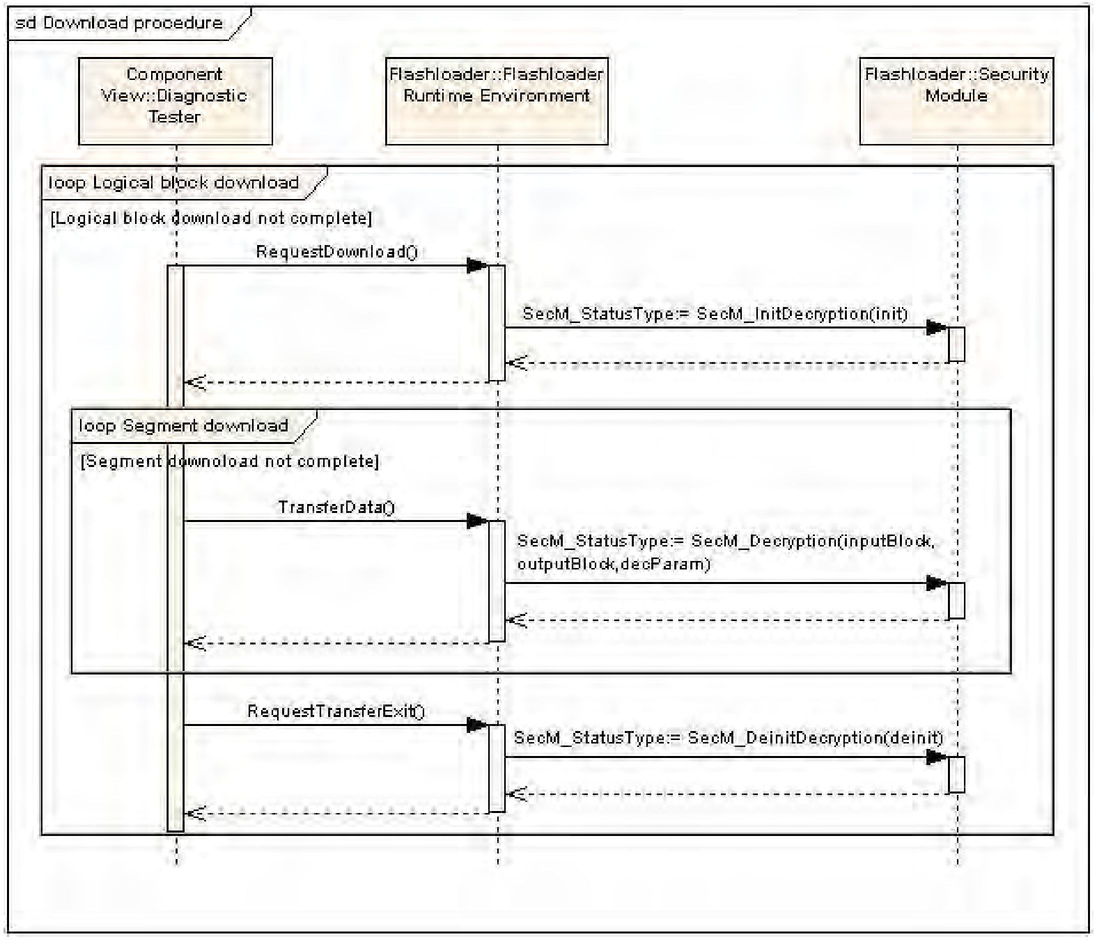

sd Download procedure
Component View::Diagnostic Tester
Flashloader::Flashloader Runtime Environment
Flashloader::Security Module
loop Logical block download
[Logical block download not complete]
RequestDownload()
SecM_StatusType:= SecM_InitDecryption(init)
loop Segment download
[Segment downlooad not complete]
TransferData()
SecM_StatusType:= SecM_Decryption(inputBlock, outputBlock,decParam)
RequestTransferExit()
SecM_StatusType:= SecM_DeinitDecryption(deinit)

Figure 62 HIS Download procedure

A.5.2 ISO 15031-7

The International Standard ISO 15031 provides a self-consistent set of specifications to facilitate emission-related diagnostics. Its part 7 [27] provides a recommendation for vehicle manufacturers for protecting vehicle modules from *unauthorized* intrusion through a vehicle diagnostic data link. The recommendation is general and can be adapted as seen in [45] to specific needs. [27] is divided into two categories: technical requirements and functional requirements.

Technical requirements In [27] an access control to product-specific software of an ECU by itself is recommended. The expected advantage is the possibility for the software to protect itself and the rest of the vehicle control system from unauthorized intrusion. The access control is realized with different seed/key combinations, which can be incorporated in any communication protocol. The standard recommends three parameters, which control the security access to the on-board controller and the secured tool:

1. Seek and key shall each be a minimum of 2 bytes in length. More bytes can be used for higher level of security. Specification of the relationship between the seed and key is left to the vehicle manufacturer. It is recommended to keep the disclosure of

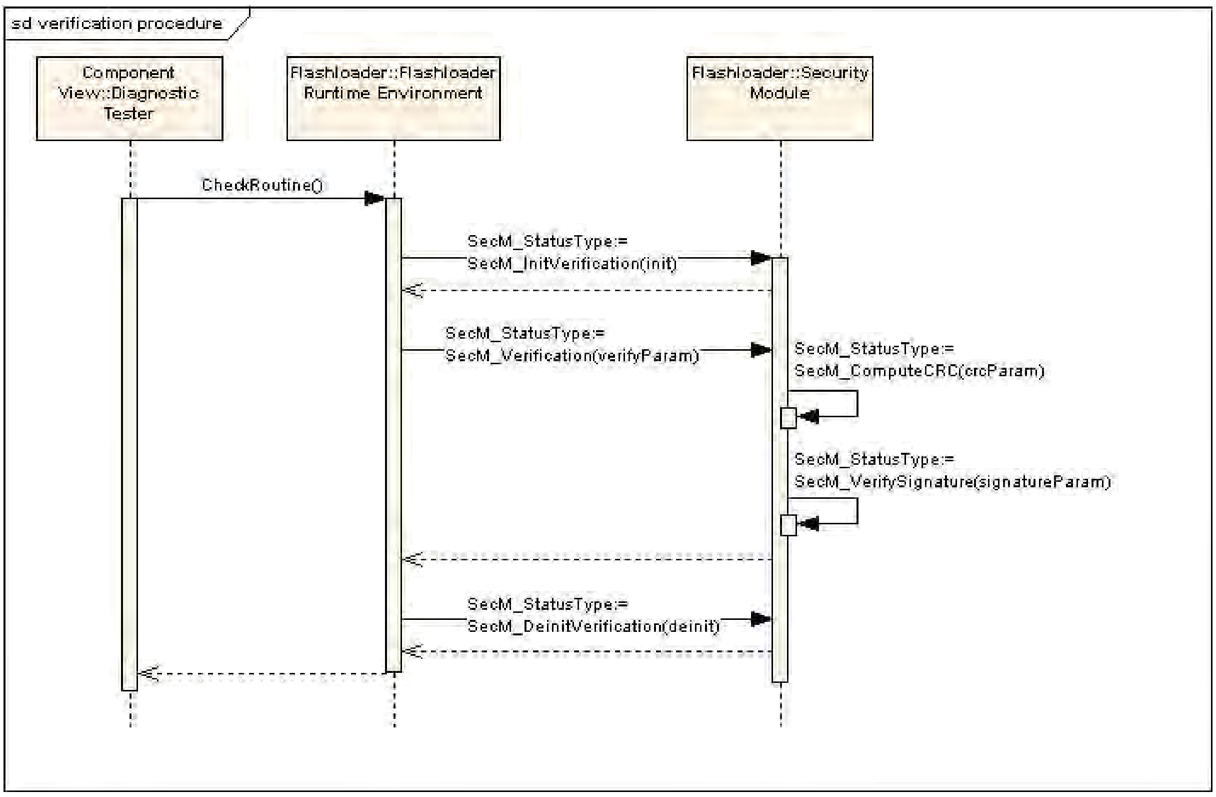

sd verification procedure
Component View::Diagnostic Tester
Flashloader::Flashloader Runtime Environment
Flashloader::Security Module
CheckRoutine()
SecM_StatusType:= SecM_InitVerification(init)
SecM_StatusType:= SecM_Verification(verifyParam)
SecM_StatusType:= SecM_ComputeCRC(crcParam)
SecM_StatusType:= SecM_VerifySignature(signatureParam)
SecM_StatusType:= SecM_DeinitVerification(deinit)

Figure 63 HIS Verification Procedure

the relationship to the persons authorized by the vehicle manufacturer.

2. Limitation of false access attempts to 2 times. A delay time is inserted if the maximum number of attempt is reached. This delay is only added if the false access attempt is related to an incorrect key.

3. Minimum of delay time is 10 seconds. This can be adapted to a larger number to suit specific requirements.

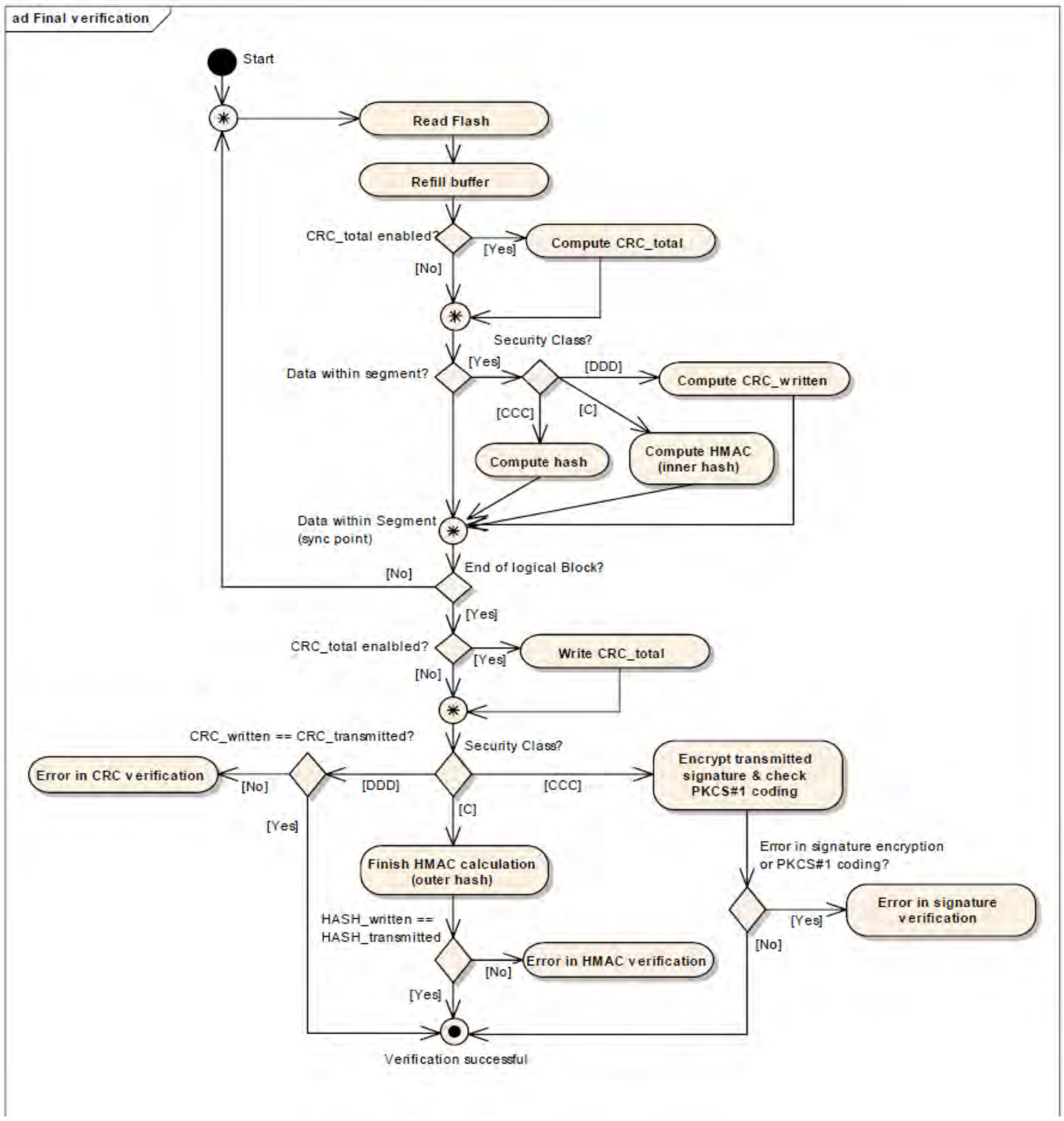
Functional requirements This paragraph specifies the steps required to grant access to the on-board controller. The standard recommends a challenge response protocol with two request/response communication message pairs to unlock the secured on-board function.

1. First the external device requests the on-board controller to unlock the desired function. The controller responds by sending a seed.

2. The external device responds by returning a key to the controller. The controller compares the key to the one internally determined and send the response back.

The standard specifies 3 types of responses:

- Accept: the controller has unlocked its access.



ad Final verification
Start
Read Flash
Refill buffer
CRC_total enabled?
[Yes]
Compute CRC_total
[No]
Security Class?
Data within segment?
[Yes]
[DDD]
Compute CRC_written
[CCC]
[C]
Compute hash
Compute HMAC (inner hash)
Data within Segment (sync point)
End of logical Block?
[No]
[Yes]
CRC_total enalbled?
[Yes]
Write CRC_total
[No]
CRC_written == CRC_transmitted?
Security Class?
Error in CRC verification
[No]
[DDD]
[CCC]
Encrypt transmitted signature & check PKCS#1 coding
[Yes]
[C]
Finish HMAC calculation (outer hash)
Error in signature encryption or PKCS#1 coding?
Error in signature verification
[Yes]
[No]
HASH_written == HASH_transmitted
Error in HMAC verification
[No]
[Yes]
Verification successful

Figure 64 HIS Verification Diagram

- Invalid key: the access attempt was rejected because the key was determined to be invalid by the controller; the access attempt was false
- Process error: the access attempt was rejected for reasons other than receiving the wrong key: this shall not be counted as a false access attempt.

The standard specifies in which condition the security access is terminated and the secured function is locked:

- Each time the controller is powered up,
- Upon commanding the product to a normal operational mode,

- Other conditions at the vehicle manufacturer's discretion.

The next section described the standard ISO 15764 which extends the security provisions of ISO 15031-7.

A.5.3 ISO 15764

The International Standard ISO 15764 [28] offers a set of procedures for enhancing the security of data transfers between ECUs and remote sources. This is intended to supplement [27] by providing more security services.

[28] documented – contrary to [27] and [45] – the specific security threats addressed by the recommended security services:

- **Masquerade**: Addressed with the entity authentication service.
- **Replay**: Addressed with the entity authentication and the data origin authentication service.
- **Eavesdropping**: Addressed with the confidentiality service.
- **Manipulation**: Addressed with the data integrity service.
- **Repudiation**: Addressed with the non-repudiation service.

[28] recommends the entity authentication service, the data integrity service, the confidentiality service as mandatory services to address the threats. Depending on the applications the non-repudiation service can be used. For audit purpose as recording or traceability an audit trail is proposed.

Following assumptions have been taken as basis:

- Secure generation, distribution and storage of keys
- Cryptographic computation are protected against malicious manipulation
- Hardware used is tamper resistant
- Used Certification authorities are known to the communication entities and are generated all the certificates used by the communication entities
- Used equipments are certified
- Used cryptographic algorithms are known and trusted by the communication entities

The recommended cryptographic algorithms are: RSA-1024, AES-128, SHA-1 and X.509 v3. [28] was published in 2004, and the recommended key length of 1024 bit for RSA algorithm is not anymore secure. Furthermore, it is not clear which MAC algorithm is recommended.

Entity authentication service For this service the standard recommends digital signatures (RSA-1024). It provides entity authentication and freshness.

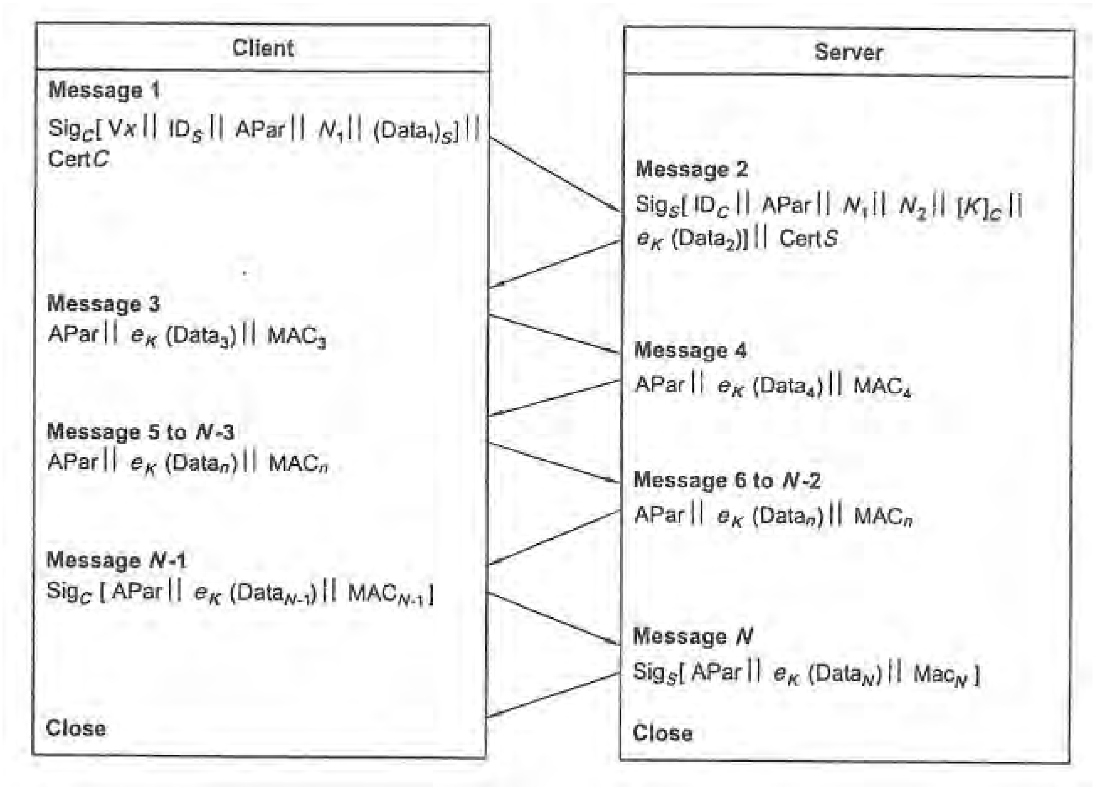

Client
Message 1
$\mathrm{Sig}_C[\,\mathrm{V}x \,||\, \mathrm{ID}_S \,||\, \mathrm{APar} \,||\, N_1 \,||\, (\mathrm{Data}_1)_S] \,||$ Cert$C$
Message 3
$\mathrm{APar} \,||\, e_K\,(\mathrm{Data}_3) \,||\, \mathrm{MAC}_3$
Message 5 to N-3
$\mathrm{APar} \,||\, e_K\,(\mathrm{Data}_n) \,||\, \mathrm{MAC}_n$
Message N-1
$\mathrm{Sig}_C\,[\,\mathrm{APar} \,||\, e_K\,(\mathrm{Data}_{N-1}) \,||\, \mathrm{MAC}_{N-1}\,]$
Close
Server
Message 2
$\mathrm{Sig}_S[\,\mathrm{ID}_C \,||\, \mathrm{APar} \,||\, N_1 \,||\, N_2 \,||\, [K]_C \,||$ $e_K\,(\mathrm{Data}_2)] \,||$ Cert$S$
Message 4
$\mathrm{APar} \,||\, e_K\,(\mathrm{Data}_4) \,||\, \mathrm{MAC}_4$
Message 6 to N-2
$\mathrm{APar} \,||\, e_K\,(\mathrm{Data}_n) \,||\, \mathrm{MAC}_n$
Message N
$\mathrm{Sig}_S[\,\mathrm{APar} \,||\, e_K\,(\mathrm{Data}_N) \,||\, \mathrm{Mac}_N\,]$
Close

Figure 65 ISO 15764:2004 Message Sequence

Confidentiality service For this service the standard recommends symmetric key cryptography (AES-128). The used keys are generated either at the start of the message exchange or are pre-shared.

Data integrity and data origin authentication service For this service the standard recommends digital signatures and MACs. The first two messages are protected using digital signatures. The subsequent messages are protected using MAC. To guarantee the integrity of the sequence of messages, each previous MAC is used as input for the next MAC. In case of pre-shared secret keys, only MAC could be used instead of MAC + digital signatures.

Non-repudiation service The standard recommends this service as optional service. Either the termination messages are digital signed, or each party requests if needed the counterpart to digital sign a message.

Message sequence Seven steps are described from the secure link setup to the end of the message transmission. Figure 65 summarizes the different steps.

[28] provides a more detailed recommendation for implementing security mechanisms for V2X communication than [27].

A.6 Cryptographic Protocols

A.6.1 Entity Authentication

Authentication is a critical function in a secure system, which is used to verify identification claims. This section deals with a first type of such claims that are related to the sender, which is referred to as entity authentication.

Overview Entity Authentication deals explicitly with the claim of some entity being a particular one or being in a particular class of entities and does not allow making any other meaningful claim. This service involves two parties: the prover and the verifier. In [63], entity authentication is defined as follows: *Entity Authentication is the process whereby one party is assured (through acquisition of corroborative evidence) of the identity of a second party involved in a protocol, and that the second has actually participated (i.e., is active at, or immediately prior to, the time the evidence is acquired.*

The goal of entity authentication is the verification of the identity of the prover. This includes the following objectives:

- Impersonation: It is not possible for a third party to take the role of the prover and to be successfully verified by the verifier.
- Real-time process: The assurance provided by the authentication process is only valid during the session, when it is executed.

In the automotive field with all the well-known constraints regarding performance and costs, the properties required of entity authentication protocols include:

- Computational efficiency: number of operations needed for the authentication process.
- Communication efficiency: number of messages exchanged. Length of the messages.
- Possibly the involvement of a trusted third party: This includes the secret management concept.
- Nature of security guarantees: The foreseen security should be provable.
- Storage of secrets: the security provided by the use of authentication protocol depends on the methods used to store the secrets.

To reach the entity authentication objectives, different entity authentication instruments are provided and can be categorized as follows:

- for the authentication of persons:
  - Something they know: This is a secret, which can be a password or a Personal Identification Number (PIN).
  - Something they possess: This can be a key, an identity card in paper form as passport, smart cards, or even hand-held customized calculators.
  - Something related to the physical characteristic of a person: e.g. fingerprint.

- for the authentication of devices:
  - cryptographic protocols.

The different categories provide different security levels in term of resistance against potential attacks. Therefore, there is another kind of categorization based on the resistance against potentials attacks: Weak authentication or passwords, toward strong authentication or one-time passwords, Strong authentication or challenge-response authentication.

Weak Authentication (passwords) This is the weakest form of authentication but still the prevalent form of human authentication to computer systems. It is based on the use of a password associated to a particular entity. This kind of authentication is unilateral. The password is stored in a system (verifier). The prover authenticates himself/herself by proving the knowledge of the password. The stored files are saved in a hashed form by applying a hash function. Password rules can also be used to make it difficult for dictionary attacks to be successful by preventing entities to use simple passwords. An alternative is the use of pass phrases, which makes it easier to remember the password, but more difficult for dictionary attacks.

Fixed passwords enable replay attacks to succeed. Beyond that they are mainly chosen in a non-large enough alphabet since the prover (usually a person) has to remember the password. This leads to vulnerabilities against exhaustive password search, password guessing or dictionary attacks.

Passwords or PINs can also be used in combination with something you possess: usually this is a chip card, e.g., a bank card. The security by the combination is more related to the chip card used and the functionalities it provides. Passkeys can also be used in combination with a fix password the user knows: the passkey is then a derived cryptographic key calculated from the password. This requires secure storage of the stored passwords.

Strong Authentication with symmetric challenge-responses Challenge-response authentication is required instead of password based techniques for ECU entity authentication. There exist two kinds of challenge-response techniques: the ones based on symmetric-key techniques and those based on asymmetric techniques.

Symmetric techniques require the use of a shared secret key. The secret key can be used for symmetric encryption or for hash functions, which leads to mechanisms based on symmetric-key encryption and mechanisms based on hash functions. [31] defines authentication protocols using symmetric cryptographic algorithms, outlined below:

2 Entities: A and B
T_A, T_B: timestamps of A and B
R_A, R_B: random number generated by A and B
E_K: symmetric encryption function
H_K: Hash function. Generally a Message Authentication Code (MAC).
B: identifier of B
A: identifier of A

- Unilateral authentication, timestamp based.
  (1) $A \to B : E_K(T_A, B)$ or
  (1) $A \to B : H_K(T_A, B), t_A, B$
- Unilateral authentication, using random numbers.
  (1) $A \leftarrow B : R_B$
  (2) $A \to B : E_K(R_B, B)$ or
  (2) $A \to B : H_K(R_B, B), B$
- Mutual authentication, using random numbers.
  (1) $A \leftarrow B : R_B$
  (2) $A \to B : E_K(R_A, R_B, B)$ or
  (2) $A \to B : R_B, H_K(R_A, R_B, B)$
  (3) $A \leftarrow B : E_K(R_B, R_A)$ or
  (3) $A \leftarrow B : H_K(R_B, R_A)$

Strong Authentication with asymmetric challenge-responses Authentication with asymmetric techniques can be done by signature creation/verification. The authenticity is then proved by demonstrating the possession of a private key corresponding to the public key used for the verification. As for challenge-response using symmetric techniques, unilateral and mutual authentications are both realizable with asymmetric techniques.

2 Entities: A and B
T_A, T_B: timestamps of A and B
R_A, R_B: random number generated by A and B
E_K: asymmetric encryption function
S: signature function
H_K: Hash function.
$Cert_A, Cert_B$: Certificate of A and B.
B: identifier of B
A: Identifier of A

- Unilateral authentication using public key decryption.
  (1) $A \leftarrow B : H_K(R_A), B, E_K(R_B, B)$ (challenge)
  (2) $A \to B : R_B$ (response)
- Unilateral authentication using digital signatures.
  (1) $A \leftarrow B : R_B$ (challenge)
  (2) $A \to B : cert_A, R_A, B, S(R_B, R_A, B)$ (response)
- Mutual authentication using public key decryption. This is a modified Needham-Schroeder PK protocol for identification.
  (1) $A \leftarrow B : E_K(R_A, A)$ (challenge)
  (2) $A \leftarrow B : E_K(R_A, R_B)$ (response and challenge)
  (3) $A \to B : R_B$ (response)

- Mutual authentication using digital signatures.
  (1) $A \leftarrow B : R_B$ (challenge)
  (2) $A \rightarrow B : cert_A, R_A, B, S(R_B, R_A, B)$ (response and challenge)
  (3) $A \leftarrow B : cert_B, R_A, A, S(R_B, R_A, A)$ (response)

Zero-Knowledge Protocols Zero-Knowledge (ZK) protocols constitute another type of protocol. They allow an entity to authenticate itself by proving that it is in possession of a secret without communicating any kind of information about itself. Those protocols require many steps and may therefore not be practicable inside vehicles. The Feige-Fiat-Shamir identification protocol or the Schnorr identification protocol are well-known protocols in that area. More on those protocols can be found in [63].

A.6.2 Data Origin Authentication

Data origin authentication (Message Authentication) in contrast to the entity authentication defined in Section A.6.1 is more concerned about the message than the sender of the message. The goals are to verify if the origin of the message content is authentic. Therefore, data origin authentication includes data integrity.

Overview In [63], data origin authentication is defined as follows: *Data origin authentication is a type of authentication whereby a party is corroborated as the (original) source of specified data created at some (typically unspecified) time in the past.*

Beyond the identification of the source origin and the data integrity, other important properties of data origin authentication are timeliness and uniqueness. Those help prevent against replay attacks and are realized by combining message authentication with time-variant parameters. The usual methods to realize data origin authentication are MACs, digital signature schemes, and secret authenticator values in combination with encryption. We will focus on the first two methods.

MACs A MAC is a key-dependent hash function. The authentication is based on the verification of a calculated hash value. To be able to calculate the hash value, a (secret) key is needed. The security of a MAC is defined as follows: It must be computationally infeasible to compute an existential forgery. Every MAC algorithm consists of three steps:

- Key generation algorithm: The output is a symmetric key with a specific length (in general 56 to 256 bit)
- MAC generation algorithm: The inputs are a message m and a key K. The output is a bit string $MAC_K(m)$ with a specific length.
- MAC verification algorithm: The inputs are the message m, the MAC value $MAC_K(m)$, the key K. The output is the result of the comparison: accept or reject.

There are two different types of MACs. Those based on block ciphers and those based on keyed hash functions.

1. MACs based on block ciphers: The CBC-MAC construction is the best-known under the MACs based on block ciphers. The CBC-MAC uses symmetric cryptographic functions and are subjects of different standardizations: One is specified in [30]. There are in general four different variants of CBC-MAC:

   - **Basic CBC MAC**: This is insecure across messages of varying lengths as stated in [63].
   - **EMAC**: A first improvement of the basic CBC MAC was proposed by the RIPE Consortium. It is secured across message of varying lengths. This has been proven by Petrank and Rackoff in [52].
   - **XCBC**: This variant just as the EMAC is secure across message of varying lengths. Furthermore, it provides computational and key management efficiency. It just requires one key for the MAC generation and an AES invocation for each block of message.
   - **OMAC**: The OMAC is an improvement of the XCBC. It is better known as CMAC. It is recommended by NIST in [15]. It reduces the key size of XCBC.

2. Keyed-Hash MACs: Keyed-Hash Message Authentication describes the MACs using hash functions instead of symmetric cryptographic functions. The de-facto-standard under those keyed hash functions are the so called HMAC constructions. They use cryptographic hash functions. The code value is here calculated by computing iteratively 2 times the hash function on message m. The first computed hash value is used together with the key as input for the second computation. The security of HMAC depends on the properties of the underlying hash function.

   H: Hash function with block length L. (E.g. SHA-1, SHA-256)
   K: secret shared Key
   ipad: inner padding. This is the byte 0x36 repeated L times
   opad: outer padding. This is the byte 0x5C repeated L times.
   $H(K \oplus opad, H(K \oplus ipad, m))$

   Currently, the most recommended HMAC is the HMAC-SHA-256 since the SHA-1 was broken.

Digital signature schemes Additionally to entity authentication, digital signatures can also be use for data origin authentication and data integrity. Furthermore, it provides non-repudiation, which is very important in case of legal issues between car manufacturers and car owners. Nowadays in the automotive world the most used signatures are RSA PKCS#1 v1.5 signatures specified in [38]:

- RSA 1024 bit: Generation: 1024 bit{1024 bit}, Verification: 1024 bit{16 bit}; Signature, private Key size: per 128 byte, public Key size $\approx$ 16 bit
- RSA 2048 bit: Generation: 2048 bit{2048 bit}, VerSig: 2048 bit{16 bit}; Signature, private Key size: per 256 byte, public Key size $\approx$ 16 bit

Because of the length of the keys, a hardware accelerator is needed in case of time constraints. Therefore, Elliptic Curve Cryptography Signature, which use smaller keys are better suited for automotive applications concerned with time constraints.

- ECDSA 160 bit (Security comparable to RSA 1024 bit): Generation: one scalar point multiplication, Verification: two scalar point multiplications: Signature length is 40 byte.
- ECDSA 256 bit (Security comparable to RSA 2048 bit): Generation: one scalar point multiplication, Verification: two scalar point multiplications: Signature length is 64 byte.

A.6.3 Key Agreement

As it cannot always be asserted that a pre-shared secret is known by any two parties (e.g., take any two hosts of the internet), public key authentication is much more popular. As encryption with asymmetric methods is of a much higher complexity (by means of processing power and thus latency), standard cryptographic protocols such as SSL/TLS are designed so that an additional *session key* is agreed upon during a handshake period, in which exclusively the asymmetric keys (which should themselves be certified by a trusted third party such as a Certificate Authority) are used.

A.6.4 Key Management

Key Distribution with Symmetric Key Infrastructure For scenarios, where only symmetric cryptography is available for a given system, there needs to be a pre-shared secret between every two communication partners. These secrets need to be communicated to the communication partners in a safe manner, e.g., in a safe environment.

If pre-shared secrets (often referred to as Pre-Shared Keys) are installed at communication partners, these secrets can be used to exchange information about keys that are changed in a regular manner, so that a breach of the current communication key does not break the overall security architecture, or is infeasible due to frequent changes of the cryptographic key material. Examples are the ANSI standard X9.17 [1] and its application [5]. This architecture generally differentiates between key-encryption (using the PSK) and message encryption (using a temporary key) as discussed in [56]. A wide-spread version which makes use of pre-shared keys and a trusted party is Kerberos, where an authentication server (AS) verifies the authenticity and grants access tickets (TGS for Ticket Granting Server, which may be independent of AS). For details, please see [48] for the specification of version 5.

Key Distribution with Asymmetric Key Infrastructure As it cannot always be asserted that a pre-shared secret is known by any two parties (e.g., take any two hosts of the internet), public key authentication is much more popular. As encryption with asymmetric methods is of a much higher complexity (by means of processing power and thus latency), standard cryptographic protocols such as SSL/TLS are designed so that an additional *session key* is agreed upon during a handshake period, in which exclusively the asymmetric keys (which may themselves be certified by a trusted third party such as a Certification Authority) are used.

A.6.5 Block Ciphers

Symmetric block ciphers take as an input a chunk of data of fixed size. A block operation then somehow combines the secret key with the input to produce an output.

The block operation usually consists of very basic operations: XOR-ing, substitution and/or permutations. Since only one block operation does not provide a reasonable level of security, several block operations are performed sequentially. In most cases, those block operations are identical but performed using different inputs: The data to encrypt / decrypt originates from the previous block operation and also the key (or part of the key) is being altered.

Neglecting the inputs, the same block operation is performed over and over again for a predefined number of repetitions. Therefore the number of repeated block operations often also is referred to as *rounds*, which is an important characteristic of a block cipher. There exist many symmetric block ciphers, but basically only two of them can be considered as well-established: 3DES and AES. The very first successful block cipher was Data Encryption Standard (DES). It features a block size of 64 bits and a key size of 56 bits. Soon the key size became the weak point of DES: With increasing performance of computational devices, a brute-force attack on the 56-bit key became more and more realistic. Triple DES (3DES) fixes this flaw by running a DES encryption/decryption three times and using a different 56-bit key each time. Until now, 3DES is considered a very secure block cipher. However, running DES three times causes 3DES to become a slow algorithm. National Institute of Standards and Technology (NIST), therefore, started a competition to find a successor for DES. Five algorithms finally remained: MARS, RC6, Rijndael, Serpent, and Twofish. Of these, Rijndael won the competition and became AES. Compared to 3DES, execution of AES is much faster. In hardware, 3DES and AES show nearly the same performance with respect to area consumption and execution speed. Due to not being a standard, only few instances of Camellia and Twofish have been implemented in hardware. Hence they are not comparable to area-optimized implementations of 3DES or AES. Even lightweight block ciphers like Iceberg or xTEA do not show a significant advantage over an area-optimized 3DES or AES. Additionally Iceberg and xTEA have not been tested for flaws that extensively.

A.6.6 Stream Ciphers

The stream cipher mode is intended to be applied whenever no additional bus load can be afforded. Especially frames sent frequently will benefit from the stream cipher mode.

Regardless of the type of frame (FlexRay, MOST, CAN, LIN), every frame consists of a certain number of payload bits p and checksum bits c. On each transmission or reception of a frame, a CSPNG generates p + c random-like bits and combines it with the payload and checksum using an XOR operation. All other bits of the frame remain unaffected. A synchronous stream cipher should be chosen, because self-synchronous stream ciphers allow an attacker to take over the communication between two nodes. Another reason a synchronous stream cipher should be chosen is that self-synchronous stream ciphers operate on blocks of data, which would increase the bus load again. Unfortunately, the level of security is not very high for the stream cipher mode. This is due to two reasons:

- Unlike cryptographic hashes, checksums are not distributed uniformly. Therefore, the chance to successfully create or modify a frame with the same encrypted checksum is higher than for hashes. Even if encrypted checksums / hashes or MACs are distributed uniformly, the chance of a collision scales with the length of the checksum / hash / MAC

- In addition, the checksums of forward error correction codes are designed to accept frames within a hamming distance of ecor. So under best conditions, the maximum level of security of the stream cipher mode is . While the reduced level of security may not be critical for CAN, FlexRay and MOST (15, 24, and 16/32 bit checksums), it may be an issue for LIN communications featuring a checksum of only 8 bits. Still the chance of an attack performed successfully is only 1/256, which might be acceptable for some applications (the LIN protocol does not include an error correction mechanism).

Summarizing, the stream cipher mode has the following advantages:

- Low latency for encryption / decryption: 1 XOR operation

- Bus load remains constant

and the following disadvantages

- Lower level of security

- Synchronization mechanism needed

A.7 Portable Consumer Devices

Mobile device and consumer electronics are increasingly integrated with on-board entertainment functionality. Besides the advantage, that mobile devices can be charged, the functionality can be seamlessly integrated to the vehicle. This means for example an integration with the HMI of the vehicle in order to control functions on the device from the vehicle and display, e.g., contacts and media information within the HMI. Currently, different interfaces and protocols are used in order to integrate devices supporting Universal Serial Bus (USB) Mass Storage Class, Apple devices and devices implementing the Media Transfer Protocol (MTP).

USB-MSC In case of using USB-MSC, the control of the functionality (e.g. play) and decoding is done by the respective Infotainment-System. Common supported formats are mp3, wma and aac.

With this solution, DRM protected content can be decoded. In order to display the media content in a comfortable manner, meta-information, e.g. IDv3-Tags, are read from the content. In case of attaching iPods/iPhones, the functionality, e.g. decoding, is done on the device, whereas the control is integrated with the Infotainment platform.

MTP Another approach is based on the Media Transport Protocol MTP. Analogue to the USB-MSC approach, control and decoding is done on the Infotainment platform. Devices supporting the Bluetooth Profiles Advanced Audio Distribution Profile (A2DP) and Audio/Video Remote Control Profile (AVRCP) the Advanced Audio provide comparable functionality. In order to improve the interoperability for the integration of mobile devices to the vehicle, the initiative Consumer Electronics for Automotive (CE4A) has been founded.

References

[1] Accredited Standards Committee X9. American national standard x9.17: Financial institution key management (wholesale). ANSI, 1985.

[2] D. Alur and D.L. Hill. A theory of timed automata. In *Theoretical Computer Science*, volume 126, pages 183–235. Elsevier Science, 1994.

[3] L. Apvrille, J.-P. Courtiat, C. Lohr, and P. de Saqui-Sannes. TURTLE: A real-time UML profile supported by a formal validation toolkit. In *IEEE Transactions on Software Engineering*, volume 30, pages 473–487. IEEE Computer Society, 2004.

[4] L. Apvrille et al. A UML-based environment for system design space exploration. In *13th IEEE International Conference on Electronics, Circuits and Systems (ICECS'2006)*, Nice, France, Dec 2006.

[5] D. Balenson. Automated distribution of cryptographic keys using the financial institution key management standard. *Communications Magazine, IEEE*, 23(9):41–46, sep 1985.

[6] H. Bar-El. Intra-vehicle information security framework. In *Proceedings of the 7th escar Conference*, Düsseldorf, Germany, 2009.

[7] M. Bishop. *Computer Security: The Art and Science.* Addison-Wesley, 2003.

[8] Boys, D. IFF Q and A, 2009. http://www.dean-boys.com/extras/iff/iffqa.html.

[9] AUTOSAR Consortium. AUTOSAR software architecture. http://www.autosar.org/.

[10] FlexRay Consortium. Flexray communication system protocol specification 2.1. http://www.flexray.com.

[11] LIN Consortium. LIN specification. http://www.lin-subbus.org.

[12] MOST cooperation. MOST specification. http://www.mostcooperation.com/home/index.html.

[13] J.-P. Courtiat et al. Experience with RT-LOTOS, a Temporal Extension of the LOTOS Formal Description Technique. *Computer Communications*, 23(12):1104–123, 2000.

[14] Sichere Intelligente Mobilität Testfeld Deutschland. simTD consortium. http://www.simtd.de.

[15] M. Dworkin. Recommendation for block cipher modes of operation: The CMAC mode for authentication. Special Publication 800-38B, NIST, 2005.

[16] A. Fuchs, S. Gürgens, R. Rieke, and L. Apvrille. Architecture and protocols verification and attack analysis. Technical Report Deliverable D3.4, EVITA Project, 2010.

[17] Vector Informatik GmbH. Introduction to FlexRay. http://www.vector-elearning.com/vl_einfuehrungfr_en,,378422.html?sprachumschaltung=1.

[18] A. Hergenhan and G. Heiser. Operating systems technology for converged ECUs. *Embedded Security in Cars*, 2008.

[19] S. Idrees, Y. Roudier, H. Schweppe, B. Weyl, R. El Khayari, O. Henniger, D. Scheuermann, G. Pedroza, H. Seudié, and H. Platzdasch. Secure on-board protocols specification. Technical Report Deliverable D3.3, EVITA Project, 2010.

[20] Road vehicles – Interchange of digital information – Controller area network (CAN) for high-speed communication. International Standard ISO 11898, 1993.

[21] Road vehicles – Controller area network (CAN) – Part 1: Data link layer and physical signalling. International Standard ISO 11898-1, 2003.

[22] Road vehicles – Controller area network (CAN) – Part 2: High-speed medium access unit. International Standard ISO 11898-2, 2003.

[23] Road vehicles – Controller area network (CAN) – Part 3: Low-speed, fault-tolerant, medium-dependent interface. International Standard ISO 11898-3, 2006.

[24] Road vehicles – Controller area network (CAN) – Part 4: Time-triggered communication. International Standard ISO 11898-4, 2004.

[25] Road vehicles – Controller area network (CAN) – Part 5: High-speed medium access unit with low-power mode. International Standard ISO 11898-5, 2007.

[26] Road vehicles – Unified Diagnostic Services (UDS) – Part 1: Specification and requirements. International Standard ISO 14229-1, 2006.

[27] Road vehicles – Communication between vehicle and external equipment for emissions-related diagnostics – Part 7: Data link security. International Standard ISO 15031-7, 2001.

[28] Road vehicles – Extended data link. International Standard ISO 15764, 2004.

[29] Road vehicles – Diagnostics on controller area networks (CAN) – Part 2: Network layer services. International Standard ISO 15765-2, 2004.

[30] Information technology – Security techniques – Message authentication codes (MACs) – Part 1: Mechanisms using a block cipher. International Standard ISO 9797-1, 1999.

[31] Information technology – Security techniques – Entity authentication – Part 2: Mechanisms using symmetric encipherment algorithms. International Standard ISO 9798-2, 2008.

[32] C. Jouvray, A. Kung, M. Sall, A. Fuchs, S. Gürgens, and R. Rieke. Security and trust model. Technical Report Deliverable D3.1.2, EVITA Project, 2009.

[33] E. Kelling, M. Friedewald, T. Leimbach, M. Menzel, P. Säger, H. Seudié, and B. Weyl. Specification and evaluation of e-security relevant use cases. Technical Report Deliverable D2.1, EVITA Project, 2009.

[34] J. Kim and Pai H Chou. Remote progressive firmware update for flash-based networked embedded systems. *ISLPED'09*, pages 407–412, 2009.

[35] D. Knorreck, L. Apvrille, and R. Pacalet. Fast simulation techniques for design space exploration. In *Objects, Components, Models and Patterns*, volume 33 of *Lecture Notes in Business Information Processing*, pages 308–327. Springer, 2009.

[36] K. Koscher and et.al. Experimental Security Analysis of a Modern Automobile. *In Proc. of the 31st IEEE Symposium on Security and Privacy*, May 2010.

[37] A. Kung et al. Security architecture and mechanisms for V2V/V2I. Technical Report Deliverable D2.1, Sevecom Project, 2008.

[38] RSA Laboratories. *RFC2313 – PKCS #1: RSA Encryption Version 1.5.* RSA Laboratories, Inc., Bedford, MA 01730 USA., March 1998.

[39] G. Leen and D. Heffernan. Modeling and verification of a time-triggered networking protocol. In *International Conference on Networking, International Conference on Systems and International Conference on Mobile Communications and Learning Technologies, 2006, ICN/ICONS/MCL 2006*, pages 178–188. IEEE Computer Society, 2006.

[40] P. Lieverse, P. van der Wolf, E. Deprettere, and K. Vissers. A methodology for architecture exploration of heterogeneous signal processing systems. In *IEEE Workshop on Signal Processing Systems, SiPS 99*, Taipei, China, Oct. 1999.

[41] H. Lockhart. Demystifying SAML. http://dev2dev.bea.com/pub/a/2005/11/saml.html, 2005.

[42] TTool, the TURTLE toolkit. http://labsoc.comelec.enst.fr/ttoolindexhtml.

[43] M. Pleil M. Busse. D1.2-10 data exchange concepts for gateways. Technical report, EASIS1, 2006.

[44] S.M. Mahmud, S. Shanker, and I. Hossain. Secure software upload in an intelligent vehicle via wireless communication links. In *Proc. IEEE Intelligent Vehicles Symposium*, pages 588–593, 2005.

[45] T. Miehling, B. Kuhls, H. Kober, H. Chodura, and M. Heitmann. HIS security module specification version 1.1. Technical Report Version 1.1, HIS Consortium, 2006.

[46] T. Miehling, P. Vondracek, M. Huber, H. Chodura, and G. Bauersachs. HIS flashloader specification version 1.1. Technical Report Version 1.1, HIS Consortium, 2006.

[47] M. Naughton, D. Hefferman, and G. Leen. Use of timed automata models in the design of real-time control network elements. In *IEEE Conference on Emerging Technologies and Factory Automation, ETFA '06*, pages 433–436. IEEE Computer Society, 2006.

[48] C. Neuman, T. Yu, S. Hartman, and K. Raeburn. The Kerberos Network Authentication Service (V5). RFC 4120 (Proposed Standard), July 2005. Updated by RFCs 4537, 5021.

[49] D.K Nilsson and et.al. Key management and secure software updates in wireless process control environments. *WiSec 08*, 2008.

[50] D.K. Nilsson and U.E. Larson. Secure firmware updates over the air in intelligent vehicles. In *Proc. IEEE International Conference on Communications Workshops ICC Workshops '08*, pages 380–384, 2008.

[51] D.K. Nilsson, Lei Sun, and T. Nakajima. A framework for self-verification of firmware updates over the air in vehicle ECUs. In *Proc. IEEE GLOBECOM Workshops*, pages 1–5, 2008.

[52] E. Petrank and C. Rackoff. *CBC MAC for real-time data sources 315-338.* Journal of Cryptography 3(13), 2000.

[53] M. Rahmani and et.al. A novel network architecture for in-vehicle audio and video streams. In *IFIP – BcN*, 2007.

[54] The RTL toolkit. http://www.laas.fr/RT-LOTOS/index.html.en.

[55] A. Ruddle, D. Ward, B. Weyl, S. Idrees, Y. Roudier, M. Friedewald, T. Leimbach, A. Fuchs, S. Gürgens, O. Henniger, R. Rieke, M. Ritscher, H. Broberg, L. Apvrille, R. Pacalet, and G. Pedroza. Security requirements for automotive on-board networks based on dark-side scenarios. Technical Report Deliverable D2.3, EVITA Project, 2009.

[56] B. Schneier. *Applied Cryptography.* John Wiley and Sons, second edition edition, 1996.

[57] E. Schoch, F. Kargl, T. Leinmüller, and M. Weber. Communication patterns in VANETs. *IEEE Communications Magazine*, 46 (11):100–109, 2008.

[58] Secunet. Towards a secure automotive platform. White paper, secunet, 2009.

[59] SEIS. SEIS consortium. http://www.eenova.de/projekte/seis/.

[60] A. Shabtai, Yuval Fledel, Uri Kanonov, Yuval Elovici, and Shlomi Dolev. Google Android: A State-of-the-Art Review of Security Mechanisms, 2009.

[61] Federal Information Processing Standards. *The Key-Hash Message Authentication Code (HMAC).* FIPS Publication 198, Federal Information Processing Standards, March 2002.

[62] Technical Surveillance Counter Measures (TSCM). IFF Basics, 2009. http://www.tscm.com/iff.pdf.

[63] A.J. Menezes P.C. van Oorschot and S.A Vanstone. *Handbook of Applied Cryptography*. CRC Press, 1996.

[64] B. Weyl, M. Wolf, F. Zweers, T. Gendrullis, M. S. Idrees, Y. Roudier, H. Schweppe, H. Platzdasch, R. El Khayari, O. Henniger, D. Scheuermann, A. Fuchs, L. Apvrille, G. Pedroza, H. Seudié, J. Shokrollahi, and A. Keil. Secure on-board architecture specification. Technical Report Deliverable D3.2, EVITA Project, 2010.

[65] W. Zimmermann and R. Schmidgall. *Bussysteme in der Fahrzeugtechnik*. Vieweg Verlag, 2004.