TUM

TECHNISCHE UNIVERSITÄT MÜNCHEN
INSTITUT FÜR INFORMATIK

SEIS AP 4.3 - Sicherheit auf Ebene der Applikation und ihrer Middleware

Alexandre Bouard, Benjamin Glas, Anke Jentzsch, Alexander Kiening, Thomas Kittel, Franz Stadler, Benjamin Weyl

TUM-I1215

Ein Projekt der
ENOVA
Innovationsallianz Automobilelektronik

GEFÖRDERT VOM
Bundesministerium
für Bildung
und Forschung

Sicherheit auf Ebene der Applikation und ihrer Middleware

Version 1.0.0, 02.02.2012

Editoren: Alexandre Bouard, Benjamin Glas

Beiträge: Alle Mitarbeiter des SEIS-Arbeitspakets 4.3

Verantwortlichkeiten:

Editor	Partner
Alexandre Bouard	BMW Group Forschung und Technik GmbH
Benjamin Glas	Robert Bosch GmbH
Anke Jentzsch	Volkswagen AG
Alexander Kiening	Fraunhofer Research Institute AISEC
Thomas Kittel	TU-München
Franz Stadler	Continental Automotive GmbH
Benjamin Weyl	BMW Group Forschung und Technik GmbH

Freigabe:

Status	Version	Datum	Freigegeben durch
Freigegeben	1.0.0	09.07.2012	

Unterschriften:

Editor: ....
Datum Name Unterschrift

Freigabe: ....
Datum Name Unterschrif

Versionen:

Version	Datum	Freigegeben durch
1.0.0	02.02.2012	Alle Mitglieder AP 4.3, Editoren: Alexandre Bouard, Benjamin Glas

INHALTSVERZEICHNIS

ABBILDUNGSVERZEICHNIS

TABELLENVERZEICHNIS

ABKÜRZUNGSVERZEICHNIS

ACL	Access Control List
AMM	Authentication Management Module
API	Application Programming Interface
C2X	Car-to-X
CCM	Communication Control Module
CE	Consumer Electronics
CRL	Certificate Revocation List
CRS	Cryptographic Services
CSM	Cryptographic Service Modules
DB	Datenbank
DoS	Denial of Service
EAM	Entity Authentication Module
ECR	ECU Configuration Register
ECU	Electronic Controller Unit
EVITA	E-safety Vehicle Intrusion proTected Applications
GPS	Global positioning Satellite
HMI	Human-Machine Interface
HSM	Hardware Security Module
HW	Hardware
ID	Identität
IDS	Intrusion Detection System
IMM	Intrusion Management Module
IP	Internet Protocol
IPS	Intrusion Prevention System
IPSec	Internet Protocol Security
ISO	International Organization for Standardization
KD	Key Distribution
KMM	Key Management Module
MW	Middleware
OBD	On Board Diagnosis
OS	Operating System
OSI	Open Systems Interconnection
PEP	Policy Enforcement Point
PDM	Policy Decision Module
PDP	Policy Decision Point
PIM	Platform Integrity Module
PKI	Public Key Infrastructure

PMM	Policy Management Module
PSK	Pre-Shared Key
Pub/Sub	Publish/Subscribe Service
PUF	Physical Unclonable Function
RPC	Remote Procedure Call
SAML	Security Assertion Markup Language
SCM	Secure Channel Module
SD	Service Discovery
SEIS	Sicherheit in Eingebetteten IP-basierten Systemen
SSL	Secure Socket Layer
SSM	Secure Storage Module
SW	Software
SWD	Secure Watchdog Module
TLS	Transport Layer Security
TPM	Trusted Platform Module
USB	Universal Serial Bus
XACML	eXtensible Access Control Markup Language

1. EINFÜHRUNG

Das AP 4.3 „Sicherheit auf Ebene der Applikation und ihrer Middleware“ hatte die Aufgabe, eine Sicherheitsarchitektur für eine in AP 3.1 [31] definierte IP-basierte Fahrzeug-Middleware vorzuschlagen. Auf der Grundlage der in AP 1.3 [16] erarbeiteten Anforderungen wurde ein Security-Framework für eine flexible Verwendung von Sicherheits-Funktionalitäten und Fahrzeug-Middleware-Diensten entworfen. Dieses Framework muss sichere IP-basierte Kommunikation von Steuergeräten, deren Spezifikationen in AP 4.2 [14] definiert werden, unterstützen. Zudem muss diese Architektur die geeignete Sicherheitsabsicherung bieten, um externe Fahrzeugdienste, die das AP 4.4 [20] definiert hat, zu integrieren.

Das Sicherheitszonenmodell von SEIS, das in AP 4.1 [17] erarbeitet wurde, definiert unterschiedliche Klassen von Sicherheitsanforderungen. Die verschiedenen Anwendungen und deren unterschiedliche Security-Anforderungen motivieren eine Einteilung in solche Sicherheitszonen. Jeder Zone wird dabei eine Sicherheitsstufe zugeordnet. Zonen mit einer hohen Sicherheitsstufe sind von Zonen mit niedriger Sicherheitsstufe klar zu separieren und an dedizierten Zonenübergängen abzusichern. Dabei gilt es, sichere – beispielsweise integere, authentische und ggf. vertrauliche – Kommunikationskanäle zu ermöglichen sowie die Kommunikation an Zonenübergängen zu filtern. Entsprechend sind zonenspezifische Sicherheitsmechanismen zu konfigurieren und umzusetzen. Diese zonenspezifischen Sicherheitskonfigurationen können in Form von Security-Plug-Ins, Schlüsselmaterial und Regeln entweder statisch vorkonfiguriert werden oder während der Laufzeit dynamisch zugeordnet werden.

Innerhalb des EU-Forschungsprojektes EVITA [11] wurde eine Sicherheitsarchitektur betrachtet, die modulare Sicherheitsdienste anbietet, die an die Bedürfnisse der Sicherheitszonen innerhalb eines Fahrzeugbordnetzes angepasst und konfiguriert werden können [Referenzen]. Diese Dienste bieten dem Steuergeräte- und Anwendungsentwickler zonenübergreifend einheitliche Schnittstellen an, z.B. über einen Security-Abstraction Layer in der Kommunikationsmiddleware (siehe Abbildung 1). Die Sicherheitsschnittstellen abstrahieren Sicherheitsdienste und kapseln, soweit möglich, konkrete Sicherheitsmechanismen, die als sogenannte Plug-Ins realisiert werden. Neben der einheitlichen zonenübergreifenden Integration von Sicherheitsmechanismen über die vorgegebenen Schnittstellen hat eine solche modulare Sicherheitsarchitektur den Vorteil, dass Sicherheitsmechanismen bedarfsgerecht durch die Wahl geeigneter Plug-Ins für die jeweiligen Zonen konfiguriert werden können, die Schnittstellen zur Applikation für den Entwickler aber gleich bleiben. Des Weiteren können Sicherheitsmechanismen durch den Plug-In-Mechanismus einfach getauscht werden, ohne dass auf Anwendungsebene Änderungen stattfinden müssen. Gerade in Anbetracht von immer wieder auftretenden Sicherheitslücken in Standardprotokollen ist eine solche modulare Sicherheitsarchitektur sinnvoll, die die Unabhängigkeit der Anwendungssoftware von den genutzten Sicherheitsmechanismen abstrahiert und damit die Austauschbarkeit vereinfacht.

Die Kommunikationsmiddleware bietet den Applikationen einheitliche Sicherheitsschnittstellen über den Security-Abstraction Layer an. Der Security-Abstraction Layer greift auf die Sicherheitsmodule zu, die vom Security-Framework angeboten werden. Die Sicherheitsmodule können zonenspezifisch konfiguriert werden und mit entsprechenden Plug-Ins ausgerüstet werden, die konkrete Sicherheitsmechanismen implementieren. Werden Sicherheitsmodule auch auf den ISO/OSI Schichten 2 bis 4 benötigt, z.B. zum Aufbau eines sicheren Kanals, so können Funktionen des Security-Frameworks auch direkt aufgerufen werden.

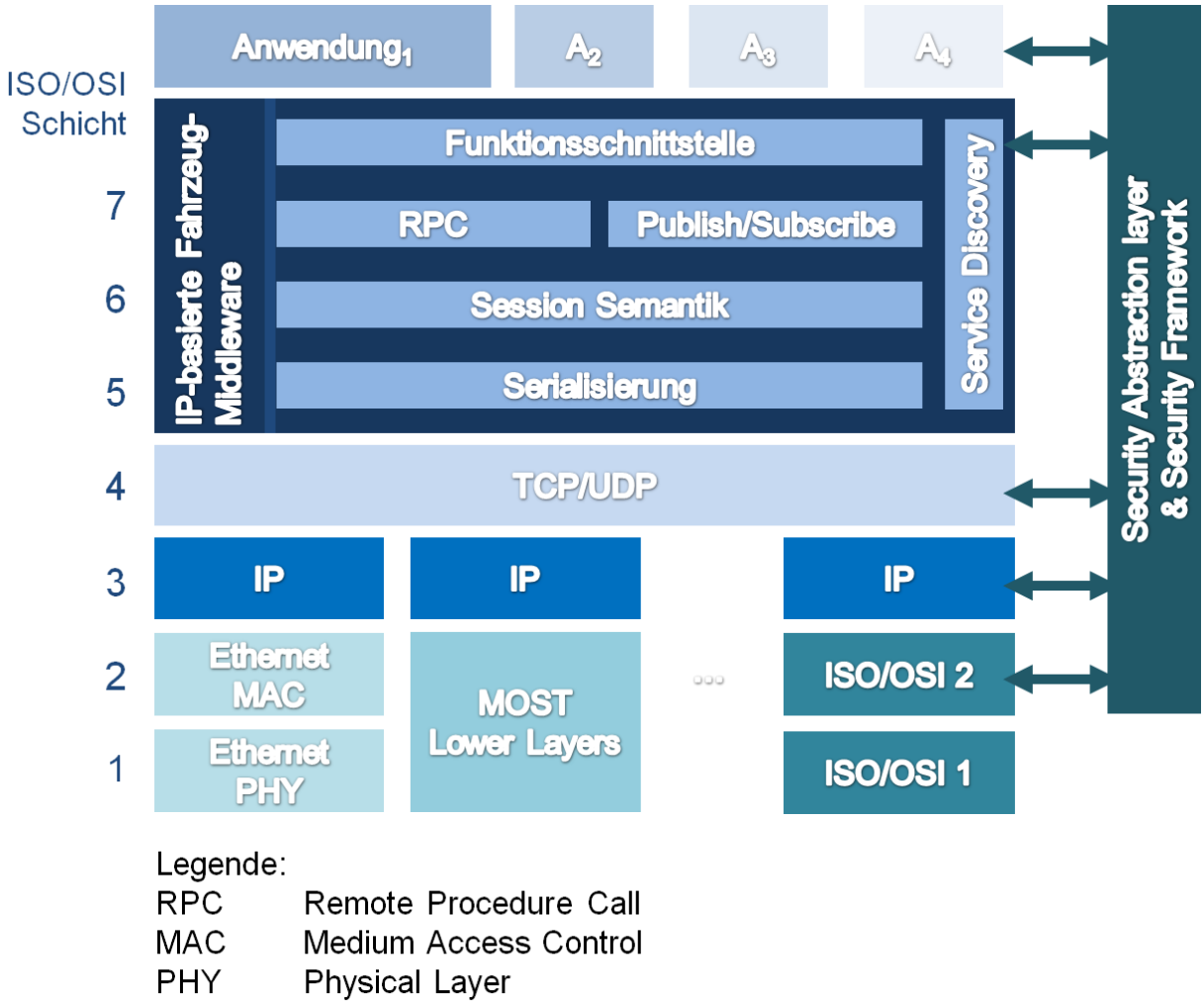

Abbildung 1: Übersicht der Middleware und Anbindung des Security Frameworks [31]

Um eine flexible und skalierbare Security-Architektur zu erreichen, sind für das Security Framework innerhalb der Middleware folgende Basiskonzepte interessant:

- Modularisierung von Sicherheitskomponenten: Durch einen modularisierten Aufbau der Sicherheitsarchitektur kann das System für spezifische Sicherheitsanforderungen angepasst und konfiguriert werden und somit für das Zielsystem optimal ausgelegt werden.
- Austauschbarkeit: Durch geeignete Modularisierung der Sicherheitsarchitektur und dem Anbieten geeigneter Schnittstellen und Plattformen steigen die Möglichkeiten der Austauschbarkeit sowie zur Erweiterbarkeit.
- Abstraktion von Schnittstellen der Sicherheitsdienste: Um die Integration von Sicherheitsdiensten, z.B. auf Anwendungsebene, zu vereinfachen, ist eine geeignete Abstraktion der Schnittstellen zu definieren.
- Konfigurierbarkeit: Statische und dynamische Konfigurierbarkeit von Sicherheitsmodulen, um neben der Vorkonfiguration die Konfiguration auch anwendungsspezifisch im Feld und während der Laufzeit dynamisch aktualisieren zu können.

Abbildung 2 zeigt die Architektur des Security Frameworks sowie die darin beinhalteten Sicherheitsmodule:

- Crypto-Services Module (CSM – Beschreibung im Kapitel 4)
  Das CSM bietet dem System kryptographische Services an. Die für die Berechnungen nötigen

Schlüssel sind in Form von Handles bereitzustellen, die zurvor vom KMM anzufordern sind. Das CSM bildet mit dem KMM eine gesonderte vertrauenswürdige Zone, die es dem CSM erlaubt direkt auf die im KMM gespeicherten Schlüsselmaterialien zuzugreifen und entsprechend dem erlaubten Zweck zu verwenden.

- Secure Communication Module (SCM – Beschreibung im Kapitel 5)
  Über dieses Modul werden sichere Verbindungen zwischen Steuergeräten aufgebaut und bedient. Dieses Modul dient als zentrale Anlaufstelle für ECU-übergreifende Kommunikation und führt die hierfür nötigen Prozeduren und Modul-Interaktionen transparent durch. Dies schließt die Überprüfung der Verbindungs-Policies beim PMM, die Authentifizierung durch das AMM und das konkrete Absichern der Verbindung mit dem CSM ein.
- Authentication Management Module (AMM – Beschreibung im Kapitel 6)
  Dieses Modul realisiert Authentifizierungsprozesse von Entitäten beim Aufbau von Kommunikationsbeziehungen. Das schließt sowohl die Client-Authentifizierung gegenüber einer anderen ECU ein, als auch die Überprüfung von Authentifierungen auf Server-Seite bei eingehenden Verbindungen.
- Key Management Module (KMM – Beschreibung im Kapitel 7)
  Die Verwaltung von kryptographischen Schlüsseln übernimmt das KMM. Hier können Schlüssel in das System eingepflegt, entfernt und nach Bedarf angefordert werden. Da direkter Zugriff auf Schlüssel ein Sicherheitsrisiko darstellt, werden primär nur Handles auf Schlüssel angefordert.
- Policy Managment Module (PMM – Beschreibung im Kapitel 8)
  Das PMM dient als ECU-zentrale Verwaltungs- und Entscheidungsstelle für die Anwendung von Policies. Module oder auch Anwendungen können hier Entscheidungen über geplante Aktionen oder Parameter für die Aktionen anfragen. Das Enforcement der Policy-Entscheidungen übernimmt hierbei die jeweils anfragende Instanz. Das Policy-Anfragen sind eng mit vielen Aktionen der Module des Security Frameworks verknüpft, wodurch das Security Framework als Ganzes die Aufgabe des Policy Enforcement übernimmt.
- Intrusion Management Module (IMM – Beschreibung im Kapitel 9)
  Die Überwachung der korrekten Funktionsweise des Steuergerätes wird durch das IMM realisiert. Hier wird das Verhalten der Software ECU-zentral protokolliert und auf Auffälligkeiten untersucht. Das Protokollieren kann z.B. über das Sammeln von Log-Meldungen bei gefundenen Policy-Verstößen in den Sicherheitsmodulen geschehen. Falls Angriffe identifiziert werden, kann das IMM lokale Reaktionen einleiten oder entsprechende Informationen an ein Netz-zentrales IMM weiterleiten.

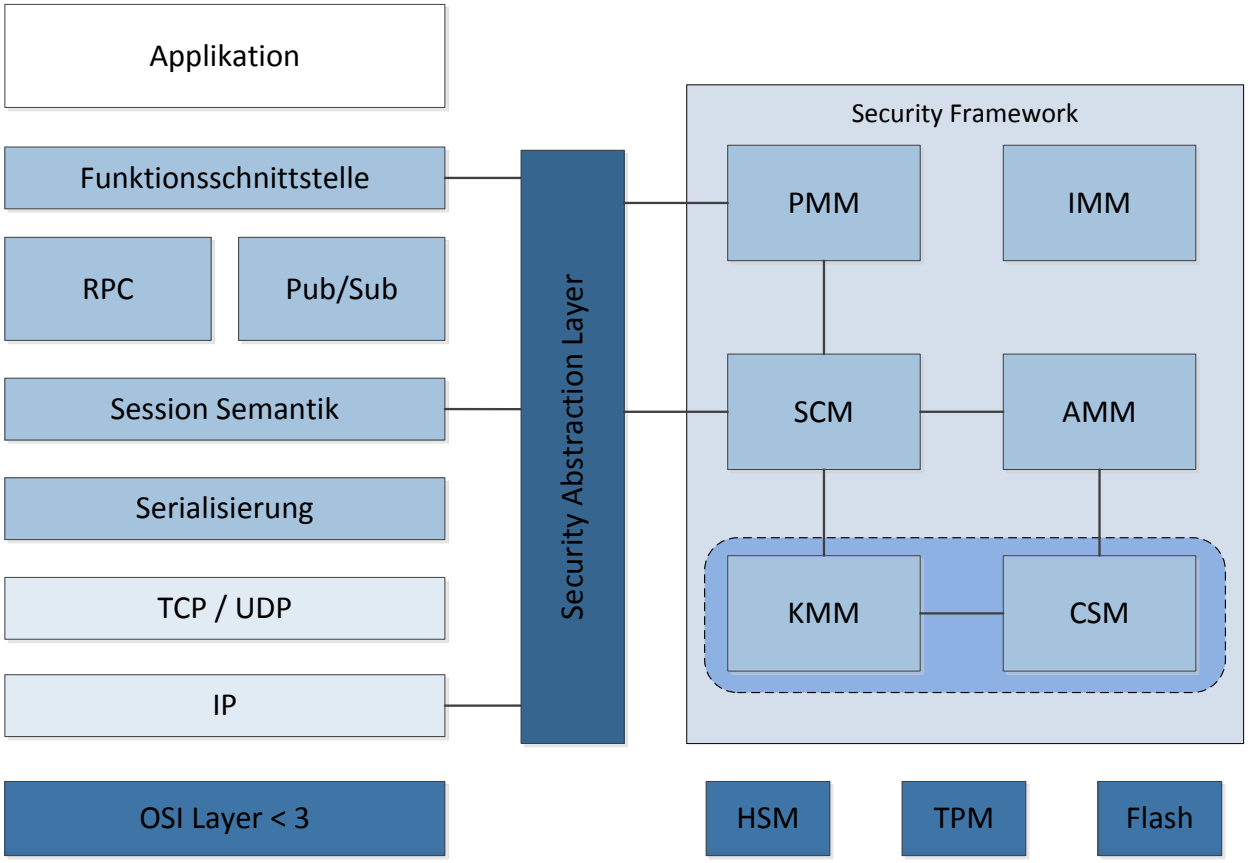

Abbildung 2: Verbindung von Middleware und Security Framework

Die Module des Security Frameworks sind sowohl Bereitsteller von Diensten als auch Konsumenten. Das AMM beispielsweise bietet Authentifizierungsdienste an, greift aber für die Durchführung auf die kryptographischen Dienste des CSM zu. Die sich durch die Aufgabenteilung der Module ergebende Verflechtung wird in Abbildung 2 veranschaulicht. Die hier gezeigten Verbindungen demonstrieren die wichtigsten Verknüpfungen der Sicherheitsmodule und sind nicht vollständig.

Zwischen den Modulen KMM und CSM herrscht eine besondere Vertrauensbeziehung. Das CSM hat für die Durchführung von kryptographischen Operationen exklusiv direkten Zugriff auf die kryptographischen Schlüssel, die im KMM hinterlegt sind. Sämtliche anderen Module können nur indirekt mit Schlüsseln über Handles arbeiten und haben keinen Zugriff auf die realen Schlüssel.

Über den Security Abstraction Layer bietet das Security Framework die Security Services dem Rest der Middleware und den Applikationen an. Grundsätzlich umfasst das die Services des SCM zum Betreiben von sicherer Kommunikation mit anderen ECUs und des PDM zum Prüfen von auszuführenden Aktionen auf deren Zulässigkeit. Der Zugriff auf die Services der Sicherheitsmodule ist nicht nur auf die beiden genannten beschränkt, sondern kann für spezielle Anwendungsfälle auch auf weitere Module gewährt werden. Ein Beispiel hierfür ist die Koppelung eines Displays mit dem IMM.zur Anzeige von Warnungen bei detektieren Sicherheitsverstößen.

Die größten Nutzer des Security Abstraction Layers in der Middleware sind die IP Schicht, die Session Semantik und die Funktionsschnittstelle. Die IP Schicht verwendet die Services des Security Frameworks zur Absicherung der Kommunikation auf IP Ebene und die Session Semantik für sichere Kommunikationskanäle auf Transport Ebene sorgt. Die Funktionsschnittstelle biete die Schnittstellen des Security Abstraction Layers der applikativen Ebene an, damit Anwendungen bei Bedarf direkt auf die Services zugreifen können.

Betrachte Anwendungsfälle werden in Kapitel 2 beschrieben. Als verwandtes Projekt wird eine Zusammenfassung und Analyse des EVITA Projektes durchgeführt. Eine ausführliche Beschreibung der Sicher-

heitsmodule findet in Kapitel 4 bis 9 statt. Eine Sicherheitsanalyse von Middleware-Dienste wird in Kapitel 10 dargestellt. Endlich wird eine Diskussion über Fehlerfälle und Sicherheit in Kapitel 11 aufgezeigt.

2. ANWENDUNGSFÄLLE

In dieser Sektion werden die Anwendungsfälle für das Ergebnisdokument aus AP4.3 beschrieben.

2.1 SOFTWARE DRAHTLOS AUS DEM NETZ HERUNTERLADEN

Über ein Zugangssystem kann mittels einer drahtlosen Anbindung Software von der Netzseite ins Fahrzeug für verschiedene Anwendungen heruntergeladen werden. Eine gegenseitige Authentifizierung ist dafür erforderlich. Dieser Anwendungsfall ermöglicht Firmware- und Flashdaten-Aktualisierung von Steuergeräte aus allen Sicherheitszonen.

2.1.1 Beschreibung

Anwendungsfall	Anwendung herunterladen und installieren
Verhalten	Anwendung auf einem Steuergerät installieren
Bedürfnis	Persönlich - Unterhaltung
Kontext	Akteur: Besitzer / Fahrer Randbedingung: Lokation mit UMTS-Signal, Fahrzeug im stehenden Zustand.
Benutzungsszenario	Der Besitzer wählt eine Anwendung aus und das Fahrzeug lädt sie von einem Back-end des Auto-Herstellers herunter und installiert sie. (z.B. Smart MP3-Player). Außer einer neuen Funktion zu installieren kann auch eine vorhandene Funktion aktualisiert werden.

Tabelle 1: Beschreibung des Anwendungsfalls "Software Herunterladen"

2.1.2 Beteiligte Entitäten

In der Abbildung 3 werden die Komponenten, die von diesem Anwendungsfall btroffen sind, dargestellt:

- Im Fahrzeug: Communication Proxy, Head-Unit, Audio Steuergerät, Display/Video Steuergerät;
- In der Umweltzone: Internet, Remote Server;

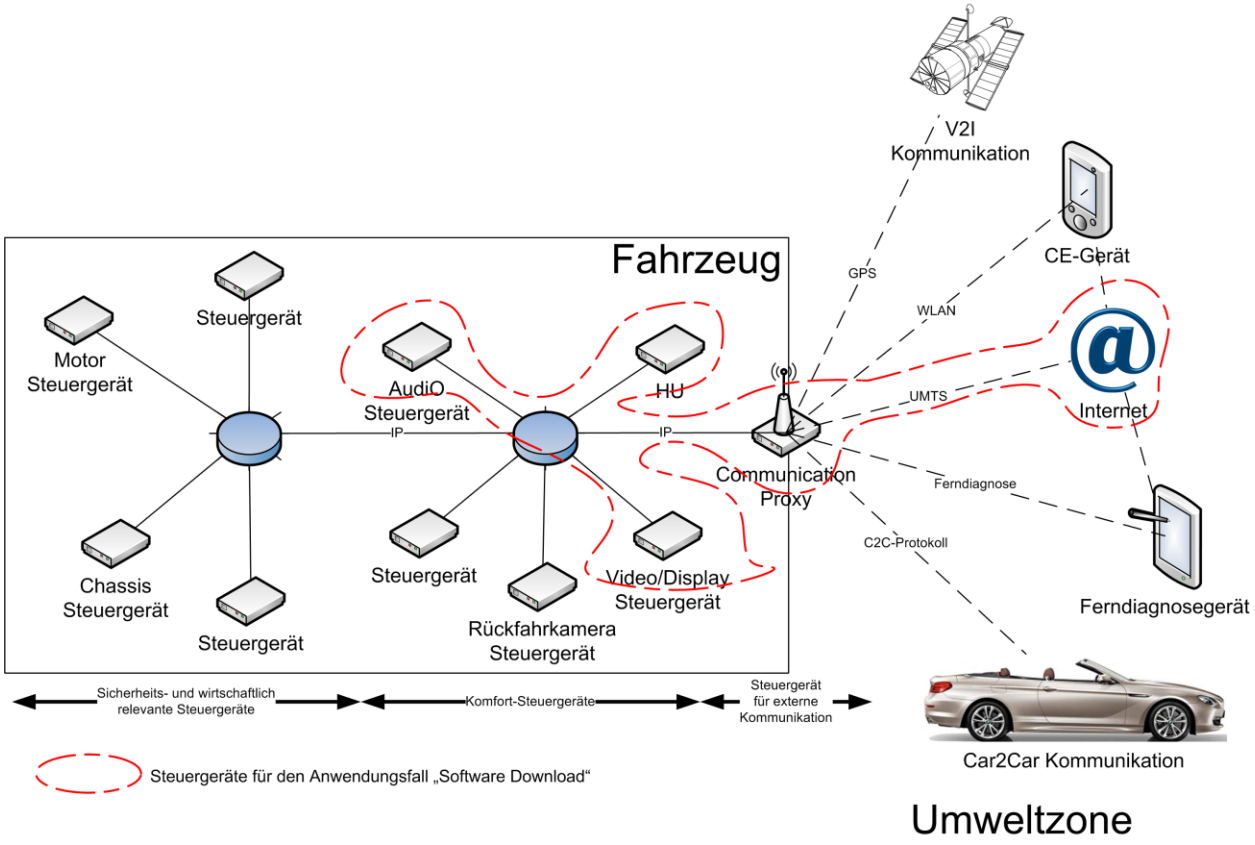

Abbildung 3: Anwendungsfall Software Herunterladen

2.1.3 Technisches Szenario

Schritt Num.	Akteur	Abnehmer	Art	Daten/Aktion
1	Benutzer	HMI	Algorithmus	Wählt den Modus “Software Download”
2	HMI	HU	Kommunikation	Schickt den Befehl “Enter Menu “Software Download”“
3	HU	-	Algorithmus	Ausführen des Befehls
4	HU	Communication Proxy	Kommunikation	Schickt den Befehl “Retrieve Application List”
6	Communication Proxy	Zertifizierter Backend Server	Kommunikation	Leitet den Befehl “Retrieve Application List” weiter
7	Server	-	Algorithmus	Ausführung des Befehls, die Liste wird generiert
8	Server	Communication Proxy	Kommunikation	Sendet die Antwort (Liste)
9	Communication Proxy	HU	Kommunikation	Leitet die Antwort (Liste) weiter
10	HU	-	Algorithmus	Verarbeitung der Antwort (Liste)
11	HU	Display/Video	Kommunikation	Sendet die Liste von Anwendungen

		Steuergerät		
12	Display/ Video Steuergerät	-	Algorithmus	Stellt die Liste dar
13	User	HMI	Kommunikation	Auswahl einer Anwendung
14	HMI	HU	Kommunikation	Sendet den Namen der Anwendung
15	HU	Communication Proxy	Kommunikation	Sendet den Befehlt “Get Application X”
16	Communication Proxy	Server	Kommunikation	Leitet den Befehl weiter
17	Server	-	Algorithmus	Holt die Anwendung X
18	Server	Communication Proxy	Kommunikation	Sendet die Anwendung X
19	Communication Proxy	HU	Kommunikation	Leitet die Anwendung X weiter
20	HU	-	Algorithmus	Ausführung des Befehls
21	HU	Display/Video Steuergerät, Audio Steuer-gerät, HU, Communication Proxy	Kommunikation	Sendet den Teil der Software zur Installa-tion
22	Display/Video Steuergerät, Audio Steuer-gerät, HU,	-	Algorithmus	Installation der Bedienung

Tabelle 2: Technisches Szenario für den Anwendungsfall "Software Herunterladen"

(Kommunikation: eine Nachricht, die aus einem Akteur zu einem Abnehmer geschickt wird;

Algorithmus: ein Prozess, der von einem Akteur durchgeführt wird;)

2.1.4 Technische Anforderungen

- Drahtlose Kommunikationsschnittstelle für das Fahrzeug (Communication Proxy für Internet-Verbindung im Fahrzeug, z.B. UMTS, LTE)
- Festverdrahtete Verbindung im Fahrzeug zwischen den Fahrzeug-Steuergeräte
- Kommunikationssystem (Middleware) im Fahrzeug zwischen den Fahrzeug-Steuergeräte
- Nicht zeitkritisch, aber eine geeignete Zeit für die Verbindung und die Installation der Anwendung soll definiert werden (User-Erfahrung)

2.2 CE-GERÄTE-INTEGRATION

Ein Fahrzeuginsasse kann für Audio/Video-Wiedergaben von Inhalten auf seinem CE-Gerät die multimedialen Fähigkeiten seines Fahrzeug-Infotainmentsystems nutzen. Er kann Audio- bzw. Videoinhalte des Fahrzeuginfotainmentsystems auf seinem CE-Geräte unter Einbeziehung des Standards UPnP nutzen.

2.2.1 Beschreibung

Use-Case	Anwendungen von CE-Geräten integrieren
Verhalten	Sichere Kommunikation zwischen Fahrzeug und CE-Gerät
Bedürfnis	Persönlich - Unterhaltung
Kontext	Akteur: Besitzer / Fahrer Randbedingung: Fahrzeug im stehenden oder fahrenden Zustand
Benutzungsszenario	Der Besitzer startet eine Anwendung auf seinem CE-Gerät. z.B. Smart MP3-Player (das CE-Gerät spielt die Titelliste des CE-Geräts auf dem Fahrzeuglautsprecher des Fahrzeug-Infotainmentsystems ab).

Tabelle 3: Beschreibung des Anwendungsfalls "CE-Gerät Integration"

2.2.2 Beteiligte Entitäten

In der Abbildung 4 werden die Komponenten, die von diesem Anwendungsfall betroffen sind, dargestellt:

- Im Fahrzeug: Communication Proxy, Head-Unit, Audio Steuergerät, Display/Video Steuergerät
- In der Außenwelt: CE-Gerät

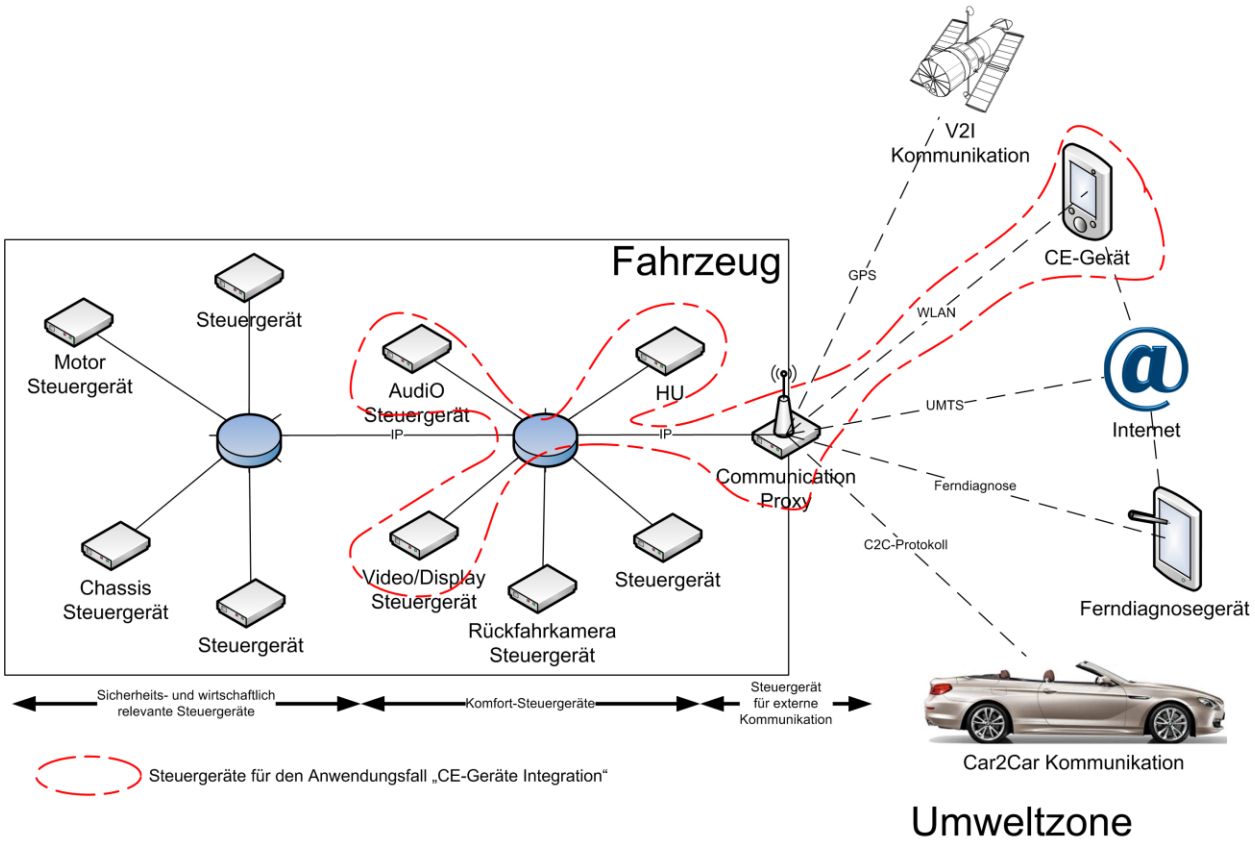

Abbildung 4: Anwendungsfall CE-Geräte-Integration

2.2.3 Technisches Szenario

Schritt Num.	Akteur	Abnehmer	Art	Daten/Aktion
1	CE-Gerät (CEG)	Communication Proxy	Kommunikation	Verbindung CEG-Fahrzeug (Authentifizierungsprozess)
2	CEG	Communication Proxy	Kommunikation	Sendet die Audio-Daten (Annahme, dass der User schon die Anwendungen für seinen CE-Gerät und sein Auto installiert hat, siehe Anwendungsfall “Software Herunterladen”)
3	Communication Proxy	HU	Kommunikation	Leitet die Daten und Befehle weiter
4	HU	-	Algorithmus	Ausführung der Befehle
5	HU	Audio Steuergerät	Kommunikation	Sendet die Audio-Daten
5	HU	Display/Video Steuergerät	Kommunikation	Sendet die Display-Daten
6	Audio Steuergerät	-	Algorithmus	Spielt die Musik ab
6	Display/ Video Steuergerät	-	Algorithmus	Zeigt die Information an

Tabelle 4: Technisches Szenario für den Anwendungsfall "CE-Gerät-Integration"

2.2.4 Technische Anforderungen

- Drahtlose Kommunikationsschnittstelle für das Fahrzeug (Communication Proxy)
- Drahtlose Kommunikationsschnittstelle für der CE-Gerät
- Fest verdrahtete Verbindung im Fahrzeug zwischen den Fahrzeug-Steuergeräte
- Kommunikationssystem (Middleware) im Fahrzeug zwischen den Fahrzeug-Steuergeräte
- Die Anfahrt dieser Funktionalität ist nicht zeitkritisch, aber eine geeignete Zeit für die Verbindung und die Ausführung der Anwendung soll definiert werden (User-Erfahrung)

2.3 REMOTE DIAGNOSE

Dieser Anwendungsfall erlaubt die Behebung von Steuergerätdefekten. Ein Fahrer kann einen Steuergerätedefekt in seiner Werkstatt beheben lassen wollen. Der Werkstattmitarbeiter hat also mit der erforderlichen Berechtigung Zugriff auf die Steuergeräte und kann nach Fehlern suchen. Das Steuergerät wird ggf. von der Werkstatt ausgetauscht.

Sonst kann ein Fahrer auch den Status seines Fahrzeugs per Remote Diagnose überprüft haben und festgestellt haben, dass die Türen nicht verschlossen sind. Er kann also die Türen per drahtloser Verbindung über die Remote Diagnoseschnittstelle verschließen. Dies erfolgt aus Sicht des Fahrers durch Anruf bei einem Call Center, über eine Webseite oder durch ein App per Mobiltelefon.

2.3.1 Beschreibung

Anwendungsfall	Remote Diagnose
Verhalten	Fahrzeugparameter auslesen und verändern

Bedürfnis	Fahrzeugwartung
Kontext	Akteur: Techniker im technischen Zentrum Randbedingung: Fahrzeug im stehenden Zustand
Benutzungsszenario	Um eine Panne zu entdecken oder den ganzen Status des Autos auszulesen, verbindet ein Techniker sein entferntes Diagnose-Gerät zum Auto. z.B. Check der Daten von der Motorleistung.

Tabelle 5: Beschreibung des Anwendungsfalls "(Remote) Diagnose"

2.3.2 Beteiligte Entitäten

In der Abbildung 5 werden die Komponenten, die von diesem Anwendungsfall betroffen sind, dargestellt:

- Im Fahrzeug : Communication Proxy, Motor-Steuergerät
- In Außenwelt: Diagnose-Gerät

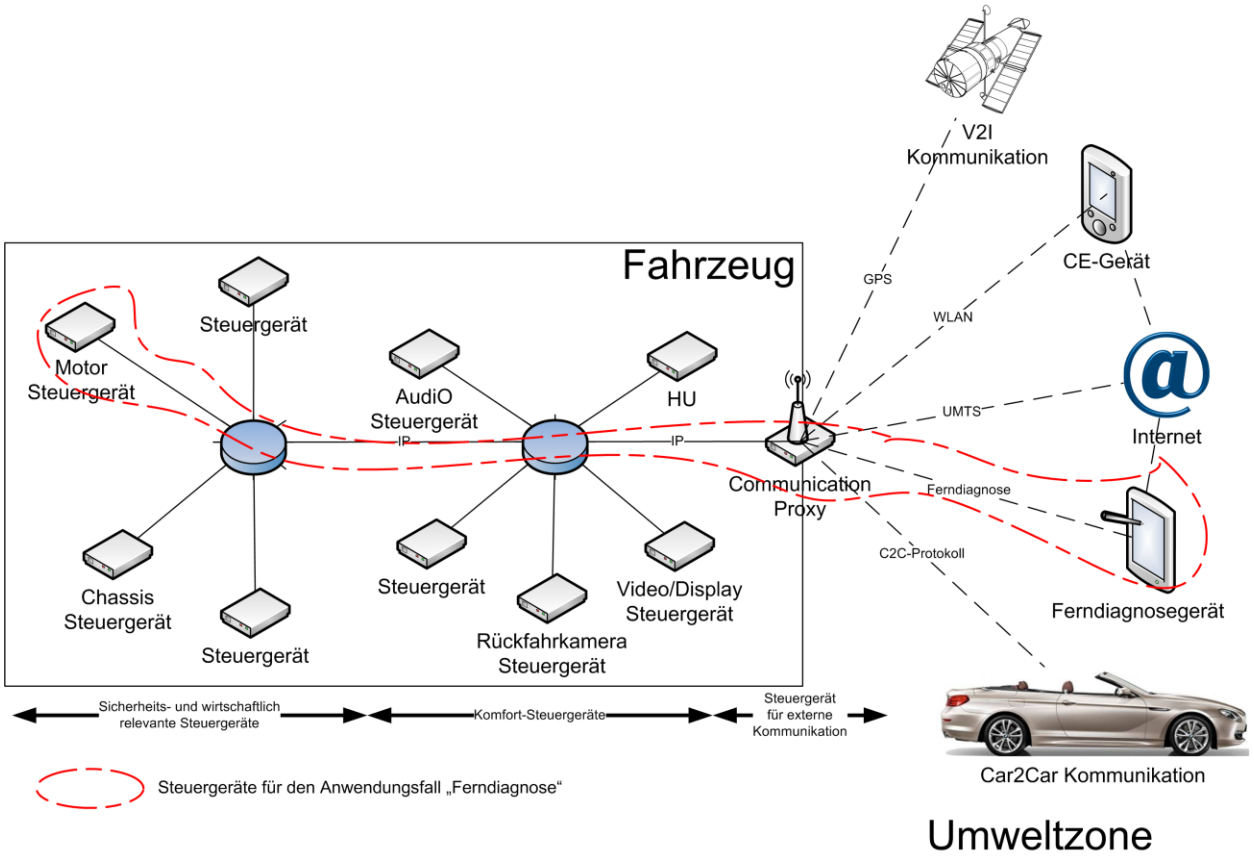

Abbildung 5: Anwendungsfall Remote-Diagnose

2.3.3 Technisches Szenario

Schritt Num.	Akteur	Abnehmer	Art	Daten/Aktion
1	Diagnose-Gerät (DG)	Communication Proxy	Kommunikation	Verbindung DG-Fahrzeug (Authentifizierungsprozess)
2	DG	Communication	Kommunikation	Sendet den Befehl “Retrieve Data X from

		Proxy		Motor ECU" (das gleiche für "Change Data X into Y in Motor ECU")
3	Communication Proxy	Motor Steuer-gerät	Kommunikation	Leitet die Nachricht weiter
4	Motor Steuer-gerät	-	Algorithmus	Ausführung der Aufforderung
5	Motor Steuer-gerät	Communication Proxy	Kommunikation	Sendet die Antwort für DG
6	Communication Proxy	DG	Kommunikation	Leitet die Nachricht weiter
7	DG	-	Algorithmus	Anzeige der Antwort

Tabelle 6: Technisches Szenario für den Anwendungsfall "Remote Diagnose"

2.3.4 Technische Anforderungen

- Drahtlose Kommunikationsschnittstelle für das Diagnose-Gerät
- Drahtlose Kommunikationsschnittstelle für das Fahrzeug (Communication Proxy)
- Festverdrahtete Verbindung im Fahrzeug zwischen den Fahrzeug-Steuergeräte
- Kommunikationssystem (Middleware) im Fahrzeug zwischen den Fahrzeug-Steuergeräte
- Nicht zeitkritisch, aber eine geeignete Zeit für die Verbindung und die Ausführung der Diagnose muss definiert werden (Techniker-Erfahrung)

2.4 RÜCKFAHRKAMERA

Dieser Anwendungsfall beschreibt das traditionelle Rückfahrkamerassystem für Parking-Assistenz.

2.4.1 Beschreibung

Anwendungsfall	Rückfahrkamera
Verhalten	Auf der zentralen Konsole des Autos wird die Sicht der Kamera nach Hinten angezeigt
Bedürfnis	Erhöhung der Fahrsicherheit (Safety) und der Fahrkomfort.
Kontext	Akteur: Fahrer Randbedingung: nach Einlegen des Rückwärtsgangs bzw. nach Betätigung eines Knopfes während des Standes oder der Fahrt
Benutzungsszenario	Wenn der Fahrer parken, oder allgemein rückwärts fahren will, kann er die Kamera Rückfahrkamera starten, um leichter Hindernisse oder Personen zu entdecken.

Tabelle 7: Beschreibung des Anwendungsfall "Rückfahrkamera"

2.4.2 Beteiligte Entitäten

In der Abbildung 6 werden die Komponenten, die von diesem Anwendungsfall betroffen sind, dargestellt:

- Im Fahrzeug: Rückfahrkamera-Steuergerät, HU, Display/Video Steuergerät;

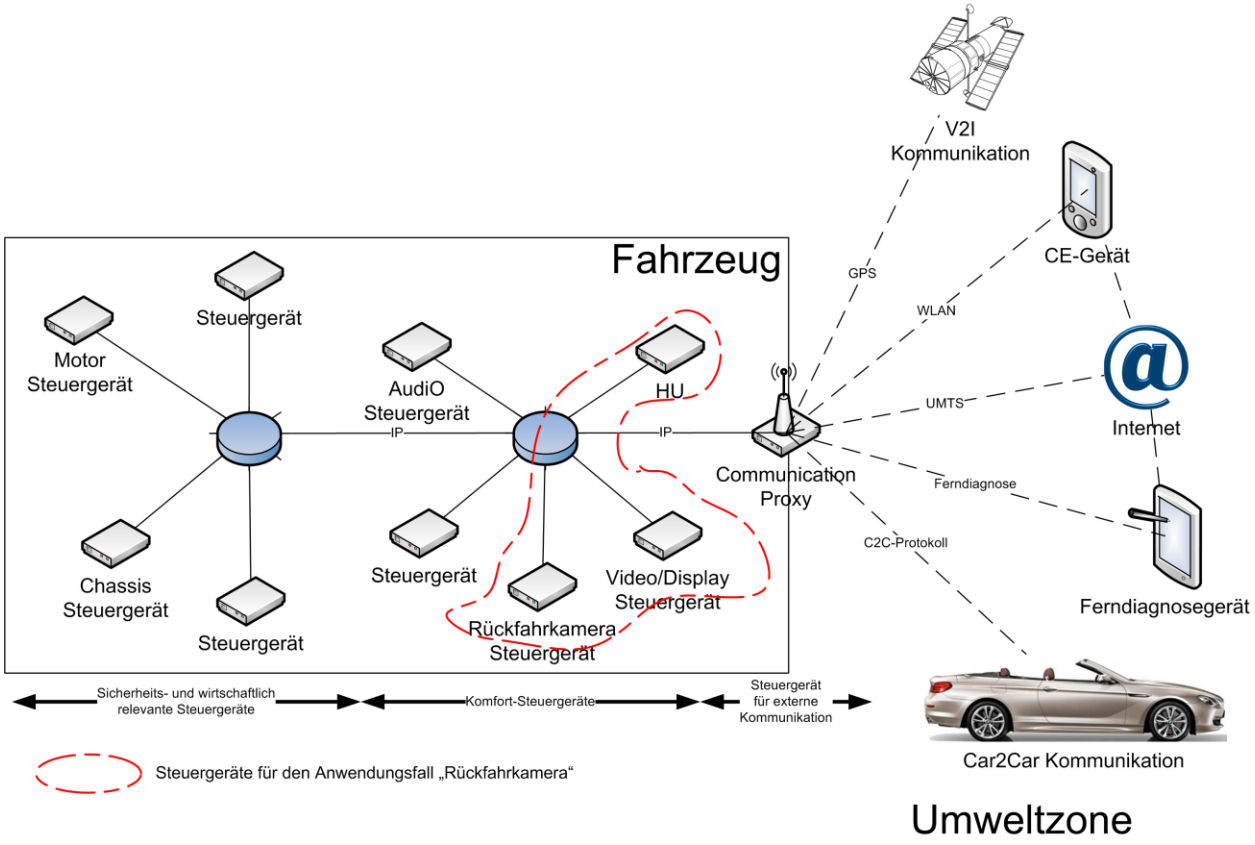

Abbildung 6: Anwendungsfall Rückfahrkamera

2.4.3 Technisches Szenario

Schritt Num.	Akteur	Abnehmer	Art	Daten/Aktion
1	User	HMI	Kommunikation	Wählt den Modus Rückfahrkamera
2	HMI	HU	Kommunikation	Sendet den Befehl “Start the Rear Camera”
3	HU	-	Algorithmus	Ausführung des Befehls
4	HU	Rückfahrkamera Steuergerät	Kommunikation	Sendet den Befehl “Get Data from the Rear Camera”
5	Rückfahrkamera Steuergerät	-	Algorithmus	Ausführung des Befehls und Bearbeitung der Bilder-Daten
6	Rückfahrkamera Steuergerät	HU	Kommunikation	Sendet die Daten
7	HU	-	Algorithmus	Verarbeitung der Daten
8	HU	Display/Video Steuergerät	Kommunikation	Sendet Video-Daten
8	Display/Video Steuergerät	-	Algorithmus	Zeigt die Sicht der Rückfahrkamera

Tabelle 8: Technisches Szenario für den Anwendungsfall "Rückfahrkamera"

2.4.4 Technische Anforderungen

- Fest verdrahtete Verbindung im Fahrzeug zwischen den Fahrzeug-Steuergeräte
- Kommunikationssystem (Middleware) im Fahrzeug zwischen den Fahrzeug-Steuergeräte
- Zeitkritisch, da Fahrsicherheitskomponente (Safety-relevant)

2.5 SICHERHEITSBEDROHUNGEN

Diese Section listet unterschiedliche Angriffe, die wir im diesem Dokument betrachten:

- Mithören der Daten-Übertragung, Informationsdiebstahl;
- Manipulation von Funktionen im Fahrzeugbordnetz durch Angriffe über die Schnittstelle für die Internet-Verbindung für:
    - Unautorisierte Zugriff auf Daten
    - Unautorisierte Veränderungen von Daten
    - Unautorisierte Kontrolle von HW-Komponenten
    - Unautorisierte Kontrolle von SW-Komponenten
    - Denial-of-Service Angriffe

2.6 SICHERHEITSANFORDERUNGEN

Diese Section stellt die Sicherheitsanforderungen dar, die das Middleware-System erfüllen muss:

Sicherheitsanforderungen	Kommentaren
Integrität	Nachrichtenintegrität: für alle ausgetauschten Nachrichten, um sicher zu sein, dass sie nicht während der Vermittlung verändert wurden. Aktualität der Daten: Nachrichten, die später erneut geschickt werden, müssen detektiert werden (Replay Attacks). Systemintegrität: Unautorisierte externe Entitäten (z.B: CE-Gerät, Remote-Server…) dürfen das System nicht verändern (Back-up Möglichkeit muss gegeben sein).
Authentifizierung	Interne und Externe Entitäten müssen vor einer Verbindung ihre Identität beweisen. Das Fahrzeugsystem muss unterschiedliche Ebenen authentifizieren: • Die Steuergeräte vom Anwendungsfall (Identität) • Die Anwendung dieser Steuergeräte • Die externe Entität (Identität) • Die Anwendung der externen Entität (Keine Entwicklung der Anwendung nach der Zertifierung des OEMs) • Den Fahrer (User-Identität) Die externe Entität muss auch unterschiedliche Ebenen von Fahrzeugkomponenten authentifizieren: • Das Fahrzeug (Identität) • Die Steuergeräte vom Anwendungsfall (Identität) • Die Anwendungen der Fahrzeug-Steuergeräte
Vertraulichkeit	• Nicht notwendig für die Kommunikation im Fahrzeug (Anwendungsfall und Angreifermodel abhängig) • Relevant für die Kommunikation zwischen Communication Proxy und externe Entität (wegen z.B. Eavesdropping)

Verlässlichkeit	Das Fahrzeugsystem muss gut gegen Angriffe geschützt sein. Gesendete Nachrichten dürfen keine DoS-Angriffe ermöglichen, die den Kommunikationskanal fluten, Einrichtungen abbauen und die Bedienung unbrauchbar machen könnten.
Nichtabstreitbarkeit (relevant für externe Kommunikation)	Relevant, um die Abstreitbarkeit von bösartigem Verhalten einer externen Entität zu verhindern, oder wenn der User die Anwendung bezahlen muss. (Log-Speicherung)
Autorisierung	Nur autorisierte Entitäten können gültige Nachricht erstellen: • Erlaubnis und Rechtsbeschränkung für die Anwendung, die in diesem Anwendungsfall eine Rolle spielt und die mit dem Fahrzeug verbunden wird (Policy Enforcement im Fahrzeug) • Erlaubnis und Rechtsbeschränkung für die Anwendungen, die am Serve • r verbunden werden (Policy Enforcement in der externen Entität)
Privatsphäre (relevant für externe Kommunikation)	Belauschte Nachrichten können nicht für Privatsphärenübergriff benutzt werden. Anonymität: Die externe Entität muss das Fahrzeug erkennen, ohne zu erlauben, dass Dritte private Information aus der Kommunikation bekommen können. • Relevant für drahtlose Kommunikation • Nicht-Relevant für Kommunikation im Fahrzeug (fest verdrahtete Entitäten)

Tabelle 9: Sicherheitsanforderungen für die vorangehenden Anwendungsfälle

3. VERWANDTES PROJEKT - EVITA

Das EU-Förderprojekt EVITA (E-safety vehicle intrusion protected applications) [11] beschäftigt sich mit der Grundlage einer gesicherten Car2Car-Kommunikation. Hierfür ist es notwendig, dass die Fahrzeuginterne Kommunikation gegen Manipulationen und Angriffe geschützt ist, damit die gesamte Nachrichtenkette vom Sensor bis zum emfangenden Fahrzeug authentisch und integer ist. Diesen gesamtheitlichen Ansatz für die Absicherung der In-Fahrzeug-Kommunikation untersucht EVITA.

Hierzu definiert EVITA eine Sicherheitsarchitektur, welche aus Hardwarebausteinen und Softwaremodulen besteht. Die Hardwarebausteine sollen folgende Merkmale besitzen:

- Hardwarebeschleunigung für bestimmte kryptographische Algorithmen
- Schutz der kryptographischen Schlüssel
- Trusted Computing Basis
- Sicherer Speicher
- Kosteneffizienz

3.1 SICHERHEITSMODULE

Die Module sind nach EVITA-Spezifikation [32] Teil der Middleware und in jedem beteiligten Steuergerät integriert.

3.1.1 EAM (Entity Authentication Module)

Charakterisierung

Das Entity Authentication Module (EAM) umfasst alle für die Authentifizierung verschiedener Entitäten notwendige Funktionalität.

Hauptfunktionalitäten des EAM sind:

- Nutzer-, Rollen- und Geräteidentifikation
- Nutzer-, Rollen- und Geräteauthentifizierung
- Nutzer-, Rollen- und Gerätemanagement im Rahmen eines Accountings
- Single Sign-On bzw. Single Sign-Off Realisierung
- Pseudonymisierung von Entitäten

Anwendbarkeit

Das EAM ist in SEIS grundsätzlich anwendbar, da auch in SEIS die Authentifizierung und Identifizierung von Entitäten durchgeführt werden muss. Es wird jedoch auch noch untersucht, inwiefern andere Architekturen zur Erbringung von AAA-Diensten (Authentifikation, Autorisierung, Accounting) wie z.B. Kerberos einsetzbar sind. Desweiteren ist unklar, wie unter Robustheitsaspekten eine sinnvolle Replizierung oder Synchronisierung von verschiedenen EAM Modulen durchgeführt werden kann.

Vor- und Nachteile

(+) Single Sign-On/Off

(+) Einheitliches Interface, Funktionalität transparent für Anwendung

(–) Eigenes Plug-In zwingend notwendig

3.1.2 PDM (Policy Decision Module)

Charakterisierung

Das Policy Decision Module (PDM) stellt Schnittstellen für Management und Zugriffsmethoden sowie für Policy Entscheidungen wie die Zugriffskontrolle zur Verfügung.

Das PDM entscheidet auf der Basis von lokalen oder verteilten Regeldatenbanken, ob der Zugriff auf eine bestimmte Ressource gewährt wird. Zusätzlich besteht die Möglichkeit, die Entscheidung an ein anderes geeigneteres PDM weiter zu geben (z.B. eine Firewall).

Die Konfiguration der Module erfolgt über die Security-Konfigurationstickets, welche zur Laufzeit verteilt werden können oder offline über Konfigurationsdateien eingebracht werden.

Bestandteile des PDM sind:

- Policy Enforcement Points (PEP)
- Policy Decision Points (PDP)
- Policy Administration Points (PAP)

Der PEP erhält die Anfragen nach Zugriff auf Ressourcen vom Sender und veranlasst die Entscheidung über die Zugriffe. Es existieren innerhalb von EVITA zwei Varianten der PEP, autonome und transparente PEP. Autonome PEP agieren als Hilfs-PDM, die auf Basis lokaler oder verteilter Regeln die Entscheidung treffen. Transparente PEP (Forward PEP) agieren als Verteiler; sie leiten die Anfragen an die PDP weiter und verteilen die Entscheidungen entsprechend. PEP in Evita verhalten sich standardmäßig wie transparente PEP.

Die PDP sind die Punkte, welche über Zugriffe auf Ressourcen auf Basis der Security Policies entscheiden. Die Policies werden von den PAP verwaltet und verteilt. Für die Kommunikation zwischen den Instanzen werden Methoden des Communication Control Module (CCM, vgl. Abschnitt [32]) verwendet.

Anwendbarkeit

Das Policy Decision Module kann in SEIS Anwendung finden. Die Mechanismen für das Policy-Handling könnten in SEIS verwendet werden. Da für die Verteilung der Policies CCM-Methoden verwendet werden, müsste entweder das CCM-Modul ebenfalls Anwendung in SEIS finden oder entsprechende auf SEIS angepasste Methoden definiert werden.

Vor- und Nachteile

(+) Online- und Offline-Verteilmöglichkeiten von Policies

(+) verteiltes Policy-Handling möglich

(+) modular aufgebaut

(–) Zeiten für die Entscheidung bei zentralem PDM können zu lang sein

(–) Ressourcenverbrauch bei zentralem PDP kann zu groß werden

Anpassungs- und Erweiterungsbedarf

Das PDM müsste auf die geplante SEIS-Sicherheitsarchitektur, insbesondere das Zonenmodell abgebildet werden. Es müsste eine Anpassung an das Sicherheitszonenmodell durchgeführt werden.

3.1.3 CCM (Communication Control Module)

Charakterisierung

Das Communication Control Module (CCM) ist dafür zuständig, die Kommunikation innerhalb von Steuergeräten (internal processes communication), zwischen Steuergeräten (in-vehicle) und mit externen Entitäten zu zu steuern und gegebenenfalls zu schützen. Hierzu gehört auch die Kommunikation zwischen unterschiedlichen Sicherheitsmodulen der EVITA-Architektur.

Hauptfunktionalitäten des CCMs umfassen:

- Kommunikationsprotokolle für verschiedene Schichten (ISO/OSI Layer)
- Paketfilter für Kommunikationsprotokolle
- Wegewahl von Dateneinheiten (Routing)
- Schutz der Kommunikation inkl. Authentizität, Integrität und Geheimhaltung

Die vom CCM dargestellten Funktionalitäten unterscheiden sich oberflächlich nicht von gewöhnlichen Kommunikationsprotokollstapeln, wie zum Beispiel von einem TCP/IP-Stack unter Linux.

Anwendbarkeit

Das CCM ist prinzipiell im SEIS-Kontext anwendbar.

Die Generalität, die das CCM benötigt, um automobile Kommunikationsmedien – wie CAN, FlexRay und MOST – zu unterstützen, ist für SEIS wahrscheinlich nicht notwendig. Es ist fraglich, ob ein Overhead durch diese Struktur zum Nachteil in SEIS werden könnte. Da sich EVITA allerdings als Baukastenlösung sieht, ist es denkbar, dass nicht notwendige Anteile weggelassen werden können und somit der Overhead signifikant gesenkt wird.

Es ist unklar, was als „Security Parameter“ definiert werden kann.

Vor- und Nachteile

(–) Unklar, welchen Mehrwert das CCM im Vergleich zu anderen Lösungen bietet

(+) Dafür jedoch AUTOSAR-Umsetzung betrachtet (siehe unten).

(+) Modularität und Flexibilität.

(–) Unklar, wie effizient das CCM sein kann, da Modularität und Flexibilität bei Kommunikationsstapeln immer auch zu stark erhöhtem Aufwand geführt haben. Es sollte jedoch auch möglich sein, eine optimierte monolithische Implementierung zu nutzen.

(+) Generisch, um LIN, CAN, FlexRay, MOST zu unterstützen

(+) Der Paketfilter-Ansatz erlaubt es, auf unterschiedlichen Schichten des ISO/OSI-Basisreferenzmodells zu filtern. Für SEIS würde man jedoch auf das übliche Vorgehen eines gemeinsamen Paketfilters für alle Schichten zurückgreifen. So wäre es möglich, zu verhindern, dass eine Dateneinheit erst auf einer höheren Schicht verworfen wird, jedoch auf tieferen Schichten zu Zustandsänderungen führt.

(–) Einige Schnittstellen sind eventuell nicht effizient implementierbar. Beispielsweise wird für den CCM::CommunicationFilter ein regulärer Ausdruck *data_filter* definiert. Üblicherweise werden für Paketfilter allerdings Ranges und Wildcards genutzt, welche sehr effizient umgesetzt werden können. Dies gilt nicht für reguläre Ausdrücke. Für eine SEIS-Lösung könnten statt regulären Ausdrücken Ranges und Wildcards genutzt werden.

(+) Protokollunabhängige Modellierung von sicheren und unsicheren Kommunikationskanälen.

(+) Implementierung in AUTOSAR betrachtet.

Anpassungs- und Erweiterungsbedarf

Es ist unklar, in wie weit bereits existierende Sicherheitsprotokolle eines Linux- oder BSD-Stacks (z.B. IPsec, TCP-MD5 und TLS/SSL) unterstützt werden.

3.1.4 SWD (Security Watchdog Module)

Charakterisierung

Das Security Watchdog Module (SWD) ist im Wesentlichen zuständig für Intrusion Detection und Prevention. Es analysiert in einer passiven wie auch aktiven Weise bestimmte Sicherheitsparameter. Des Weiteren ist die korrespondierende Reaktionsstrategie direkt oder indirekt möglich.

Hauptfunktionalitäten des SWDs umfassen:

- Selbstständige Überwachung / Anfrage von sicherheitsrelevanten Parametern (*SecurityParameterQuery_Plugins*)
- Empfang von Sicherheitsereignissen mittels der bereitgestellten Funktion *SWD_log_event()* von anderen Sicherheitsmodulen und Entitäten
- Meldung von Sicherheitsereignissen an interne (*SWD_security_response_plugins*) und externe (*SWD_listener*) Empfänger. Wobei die externen Empfänger individuell auf ein Sicherheitsereignis außerhalb von SWD reagieren können.

Anwendbarkeit

Das SWD ist mit seiner Plugin-Struktur ziemlich flexibel und generisch gehalten und somit theoretisch auch auf SEIS anwendbar. Vor allem, und das ist auch in EVITA so, dient das SWD als Plattform für Intrusion Detection und deren Reaktionen.

Vor- und Nachteile

(+) Query- und Response-Plugins modular und flexibel erweiterbar

(+) Plugins können unterschiedliche Ausprägungen darstellen e.g. Netzwerk-Monitor, Execution Monitor, usw.

(+) Selbstständige Überprüfung von Sicherheitsparametern ist flexibel und in unterschiedlichen Abständen möglich

(–) Sicherheitsparameter müssen zuvor definiert und registriert werden

(–) Ohne CCM müssen Signatur-Verifikationen für SecurityEvents durchgeführt werden, was ein Vorhalten von Zertifikaten voraussetzt.

Anpassungs- und Erweiterungsbedarf

Das Grundprinzip des SWD kann ohne größeren Aufwand auch in SEIS Anwendung finden. Generell bestehen hier zwei Möglichkeiten ein Intrusion Detection System zu integrieren. Zum einen können Sicherheitsparameter vordefiniert werden und vom SWD in regelmäßigen Abständen kontrolliert werden oder man integriert das IDS mittels der SWD_log_event() Funktion. Beim letzterem sendet das IDS im Falle eines Alarms einen Event an die SWD_log_event Funktion, welche dann wiederum entsprechend der Konfiguration darauf reagiert.

3.1.5 PIM (Platform Integrity Module)

Das Platform Integrity Module dient den nachfolgend beschriebenen vier Hauptaufgaben. Um diese sicher zu realisieren ist das PIM fest mit einem Sicherheitsanker wie bspw. einem HSM verknüpft. Die Aufgaben sind wie folgt:

- Initialisierung der Trusted Computing Base (TCB) einer Plattform
- Attestation
- „Chaining“ / „Unchaining“ von Daten zu einer Plattformkonfiguration
- Plattformidentifikation

Das PIM bearbeitet alle Schritte der Plattforminitialisierung, d.h. Verarbeitung der Integritätsmessungen der Plattform und aller Abhängigkeiten. Hierdurch kann ein sicherer, authentifizierter Bootvorgang realisiert werden, bei dem Manipulationen der Plattform erkannt werden.

Mittels der Attestationsfunktionalität können andere Entities die aktuelle Plattformkonfiguration überprüfen.

Die „Chaining“ / „Unchaining“ Funktion ist analog zum „Sealing“ / „Unsealing“ der TCG spezifiziert, d.h. Daten werden an eine bestimmte Plattformkonfiguration gebunden und Zugriff ist nur bei unveränderter Plattformkonfiguration möglich. Wenn erwünscht, kann das PIM zur eindeutigen Identifikation einer Plattform verwendet werden.

Anwendbarkeit

Die Softwarekomponente PIM ist nur in Kombination mit einem Hardwarebaustein wie dem HSM oder anderen TPM-artigen Bausteinen sicher einsetzbar. Unter der Annahme, dass das HSM oder ein ähnlicher Baustein in SEIS einsetzbar ist, ist auch das PIM einsetzbar. Die PIM-Funktionalitäten orientieren sich an vergleichbaren Funktionen des TPMs. Da im Rahmen von SEIS aber ebenfalls Szenarien betrachtet werden, bei denen nicht alle (kritischen) ECUs mit einem HSM ausgestattet sind, sind die derzeitigen Funktionen des PIM alleine nicht ausreichend. Neben den hardwarebasierten Schutzmechanismen werden deshalb im Rahmen von SEIS noch softwarebasierte Ansätze (wie z.B. softwarebasierte Attestationsprotokolle) untersucht.

Vor- und Nachteile

(+) Funktionalitäten aus dem TPM-Bereich bekannt und gut erforscht (viele existierende anwendbare Protokolle)

(+) Bietet alle benötigten Funktionen (in Kombination mit entsprechender Hardware)

(–) In SEIS zusätzliche Ansätze benötigt, die nicht auf einem HSM basieren.

Anpassungs- und Erweiterungsbedarf

Alle Funktionalitäten des PIM sollten ohne großen Anpassungsbedarf auch in SEIS verwendet werden können. Da in SEIS aber auch Szenarien betrachten werden, bei denen nicht alle (kritischen) ECUs mit einem HSM ausgestattet sind, müssen noch weitere Funktionalitäten (wie z.B. Software-basierte Attestation) ergänzt werden oder in einem eigenen Modul spezifiziert werden.

3.1.6 SSM (Secure Storage Module)

In diesem Abschnitt wird das SSM beschrieben, welches der sicheren Ablage von kritischen Daten (wie z.B. kryptografischen Schlüsseln) dient.

Charakterisierung

Das SSM spezifiziert, wie Daten sicher in einem nichtflüchtigen Speicher gespeichert und entsprechend verarbeitet werden können. Hierzu werden die Funktionen create(), open(), read(), write(), close(), und delete() spezifiziert.

Das SSM soll Schutz gegenüber Angriffen auf die Vertraulichkeit, Integrität und Authentizität der Daten bieten. Ebenfalls soll ein Schutz vor Replay-Angriffen die Aktualität der Daten garantieren und anhand von Policies der Zugriff auf Daten kontrolliert werden. Um dies zu erreichen, ist analog zum PIM ein Einsatz mit einem HSM oder TPM-artigen Hardwarebaustein nötig.

Anwendbarkeit

Analog zum PIM basiert das SSM auf den Schutzfunktionen des HSM. Unter der Annahme, dass das HSM oder ähnliche Bausteine in SEIS einsetzbar sind, ist ebenfalls das SSM einsetzbar.

Vor- und Nachteile

(+) bietet ähnliche Funktionen wie das TPM zum sicheren Speichern von Daten

(–) basiert auf der Sicherheit des HSM, Schutz gegen Manipulationen ggf. unzureichend (siehe Kapitel 3.1.7)

Anpassungs- und Erweiterungsbedarf

Alle Funktionalitäten des SSM sollten ohne großen Anpassungsbedarf auch in SEIS verwendet werden können

3.1.7 CRS (Cryptographic Services)

Die Cryptographic Services stellen ein Interface zu kryptografischen Funktionen und Primitiven zur Verfügung. Sie stellen letztlich eine Funktionsbibliothek dar, die von Security-Modulen oder –Programmen, die Kryptofunktionen benötigen, nach Bedarf genutzt werden können.

Charakterisierung

Die Cryptographic Services sind eine Sammlung zustandloser Bibliotheksroutinen. Bei ihrer Nutzung müssen Security-Merkmale, wie zum Beispiel Schlüssel, Zertifikate, Passworte oder Zufallswerte, von der sie nutzenden Applikation verwaltet und gespeichert werden. Jede Krypto-Operation erfolgt ohne Bezugnahme auf eventuell vorausgegangene Verarbeitungsschritte.

Die derzeit vorliegende Ausprägung der Cryptographic Services umfasst alle für moderne Security-Applikationen erforderlichen Kryptoverfahren und beinhaltet eine Auswahl der wichtigsten kryptografischen Algorithmen:

- Hashes: SHA1, SHA-224, SHA-256, Whirlpool, MD5

- Authentifikationscodes: AES-OMAC1, HMAC-SHA1
- Verschlüsselung
  - Verfahren: (3)DES, AES-128, TDEA, RSA-2048, ECC-Secp160r1
  - Verkettung: CBC, CFB, OFB, CTR, GCM
- Digitale Signaturen: ECDSA-SHA1-Secp160r1
- Diffie-Hellman-Schlüsselberechnung
- Kryptografisch starke Zufallszahlen

Diese Verfahren werden weitgehend durch Hardware Security Modules (HSM) unterstützt, um Prozessoren von rechenaufwändigen Operationen zu entlasten. Für verschiedene Anwendungsbereiche stehen unterschiedlich leistungsfähige HSM-Komponenten zur Verfügung.

Anwendbarkeit

Einer Nutzung der von EVITA bereitgestellten Cryptographic Services in SEIS stehen aus derzeitiger Sicht keine prinzipiellen Gründe entgegen. Voraussetzung hierfür wäre allerdings, dass auch andere EVITA-Konzepte übernommen werden, insbesondere hinsichtlich der engen Kopplung kryptografischer Dienste mit den in EVITA definierten Security-Protokollen und HSM-Komponenten.

Vor- und Nachteile

(+) Relativ vollständige und erprobte Funktionsbibliothek.

(+) Unterstützung von Krypto-Operationen durch HSM-Bausteine, sofern vorhanden.

(+) Skalierbarkeit für unterschiedlich leistungsstarke Rechnerkomponenten.

(–) Es werden nur einige ausgewählte Sicherungsstärken unterstützt, insbesondere nur eine Schlüsselstärke je Verschlüsselungsverfahren.

(–) Nutzung proprietärer Formate für kryptografische Zertifikate.

Anpassungs- und Erweiterungsbedarf

Sofern die Cryptographic Services auch für Sicherheitsfunktionen mit internetbasierten Diensten genutzt werden sollen, müssten einige dort häufig anzutreffende (Kombinationen von) Kryptoverfahren und Sicherungsstärken ergänzt werden. Wenn diese nicht durch Krypto-Operationen von HSM-Komponenten abgedeckt sein sollten, wäre eine prinzipiell laufzeitaufwändige Software-Implementierung erforderlich. Auf leistungsstarken Rechnerkomponenten, wie etwa Head Units, sollte dies jedoch unkritisch sein. Gegebenenfalls würde dies jedoch dazu führen, dass für bestimmte Algorithmen mehrere parallele Implementierungen existierten.

Für die Verwendung der Cryptographic Services mit quelloffener Software, die zum Beispiel auf Kryptofunktionen von OpenSSL zugreift, wäre zudem eine Adaption der jeweiligen Kryptobibliothek mit HSM-Funktionen zu erwägen.

3.1.8 HSM (Hardware Security Module)

Charakterisierung

In EVITA wird zwischen drei HSM-Varianten unterschieden: *light*, *medium* und *full*. Architekturell sind HSM und CPU zusammen auf einem Chip realisiert und sind programmierbar. EVITA verfolgt den Ansatz, dass HSMs in allen (kritischen) ECUs verwendet werden, um eine möglichst vollständige Absicherung zu erreichen. Die HSMs können beispielsweise folgendermaßen in einem Fahrzeug eingesetzt werden:

- *full* in 1–2 Leistungsfähigen, zentralen ECUs (z.B. Head Unit, zentrales Kommunikationsgateway)
- *medium* in 2–4 zentralen Mehrzweck-ECUs (z.B. Motorsteuerung, Front/Rear Module, Immobilizer)
- *light* in weniger sicherheitskritischen ECUs (z.B. kritische Sensoren und Aktuatoren)

In Tabelle 10 sind die Eigenschaften der drei HSM-Ansätze und ein Vergleich mit SHE-Modul, TPM und einer klassischen Smartcard zusammen gefasst.

	EVITA Full	EVITA Medium	EVITA Light	SHE	TPM	Smartcard
Boot Integrity protection	Auth. & Secure	Auth. & Secure	Auth. & Secure	Secure	Auth.	None
HW crypto algorithms (incl. Key generation	ECDSA,ECDH, AES/MAC, WHIRLPOOL/ HMAC	ECDSA,ECDH, AES/MAC, WHIRLPOOL/ HMAC	AES/MAC	AES/MAC	RSA, SHA-1/MAC	ECC, RSA, AES, 3DES, MAC, SHA-X …
HW crypto acceleration	ECC, AES, WHIRLPOOL	AES	AES	AES	None	None
Internal CPU	Programmable	Programmable	None	None	Preset	Programmable
RNG	TRNG	TRNG	PRNG w/ ext. Seed	PRNG w/ ext. Seed	TRNG	TRNG
Counter	16x64bit	16x64bit	None	None	4x32bit	Yes
Internal NVM	Yes	Yes	Optional	Yes	Indirect (via SRK)	Yes
Internal Clock	Yes w/ ext. UTC sync	Yes w/ ext. UTC sync	Yes w/ ext. UTC sync	No	No	No
Parallel Access	Multiple Sessi-ons	Multiple Sessi-ons	Multiple Sessions	No	Multiple Sessions	No
Tamper protection	Indirect (pas-sive, part of ASIC)	Indirect (pas-sive, part of ASIC)	Indirect (passive, part of ASIC)	Indirect (passive, part of ASIC)	Yes (mfr. Dep.)	Yes (active, up to EAL5)

Tabelle 10: Vergleich verschiedener HSM Ansätze

Anwendbarkeit

Grundsätzlich sollten alle drei EVITA-Module auch in SEIS einsetzbar sein. Unklar ist jedoch wie die Verwendung von IP sich auf die HSMs auswirkt. Als Technologien werden in EVITA klassische Fahrzeugbussysteme wie CAN, MOST und Flexray genannt. Durch den höheren Overhead von IP könnten die HSMs eventuell nicht mehr in der Lage sein, z.B. Daten zu Signieren und noch bestimmte Echtzeitanforderungen einzuhalten.

Weiterhin setzt EVITA in der Architektur auf einen verteilten Ansatz, bei dem alle (kritischen) ECUs mit einem HSM ausgestattet sind. Der geplante Ansatz bei SEIS ist jedoch modularer und kann von einer einzigen ECU mit HSM (die als zentraler Vertrauensanker dient) über Mischformen bis zum EVITA-Ansatz reichen. Hier ist noch zu klären, ob die in EVITA entwickelten HSMs auch in solchen Architekturen einsetzbar sind. Lässt sich beispielsweise mit einem full oder medium HSM eine zentrale Vertrauensanker-ECU realisieren?

Vor- und Nachteile

(+) Auf Basis von OEM-Anforderungen entworfen

(+) Bietet alle kryptografischen Funktionen, die in SEIS relevant sind.

(+) Angepasst an den automotive Einsatz

(–) keine Zertifizierung der Manipulationsresistenz

(–) Schutz gegen Manipulationsresistenz geringer als bei TPM oder Smartcards

(–) Einsatzmöglichkeit von IP unklar

(–) Unklar, ob auch Architekturen möglich sind, bei denen nicht alle (kritischen) ECUs mit einem HSM ausgestattet sind

(–) Hardware noch nicht verfügbar; für Demonstratoren wird die Hardware zu spät verfügbar sein, als Teil der theoretischen Architektur könnten die EVITA-HSMs aber angenommen werden

Anpassungs- und Erweiterungsbedarf

Ggf. Anpassungen zur Unterstützung von IP und entsprechenden Protokollen (z.B. IPsec) erforderlich.

3.2 PROTOKOLLE UND VERFAHREN

Evita [30] definiert neben den einzelnen Modulen ebenfalls entsprechende Protokolle und Verfahren für die Kommunikation im Fahrzeug.

3.2.1 Key Distribution

Die Schlüsselverteilung in Evita basiert auf dem Vorhandensein von Hardware Security Modulen (HSMs). Diese stellen die in Tabelle 10 aufgelisteten Fähigkeiten zur Verfügung und ermöglichen es – im Vergleich zu einer nicht hardwareunterstützten Plattform – durch zusätzliche Investition in Hardware ein erhöhtes Sicherheitsniveau herzustellen.

Charakterisierung

Zur Verteilung von Schlüsseln wird auf die in Abbildung 7 dargestellte Deployment-Referenzarchitektur von Evita zurückgegriffen, in die später bei Bedarf ein Key Master, wie in Abbildung 8 dargestellt, integriert wird .

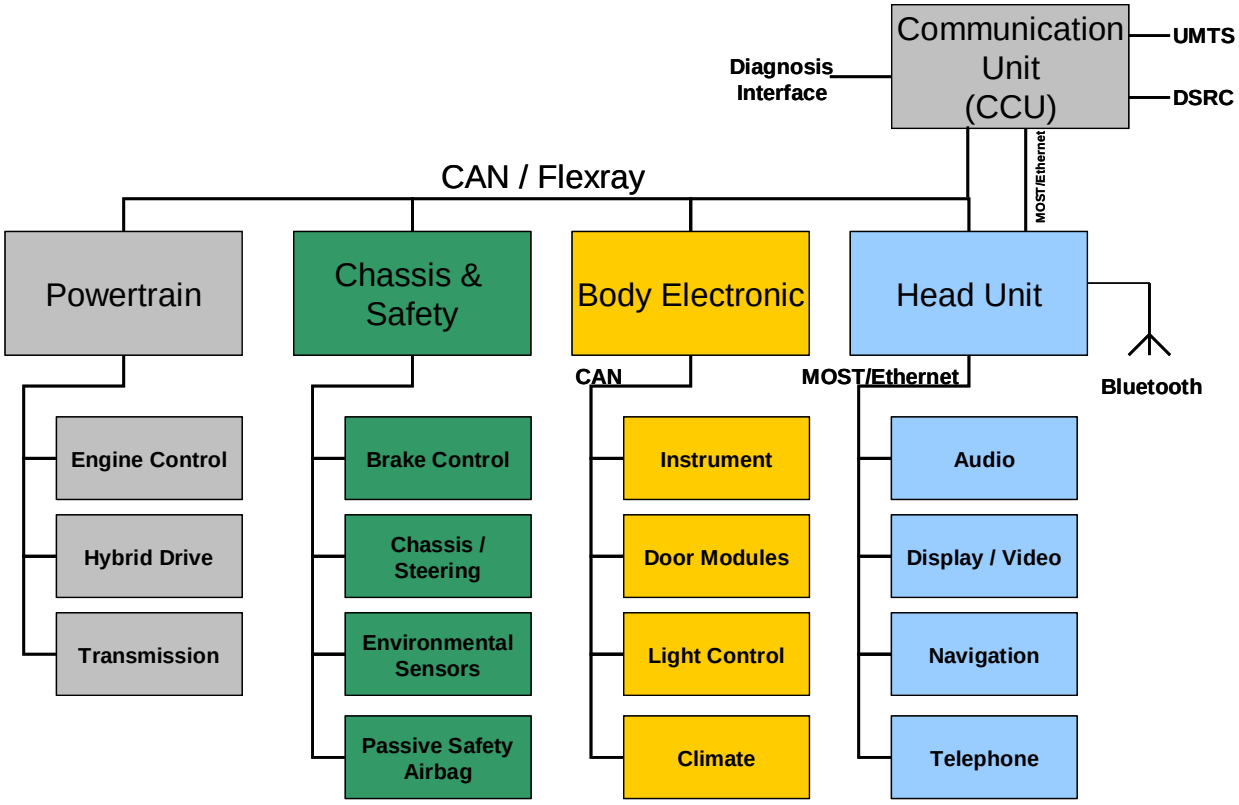

Abbildung 7: Skizze der EVITA-Referenzarchitektur

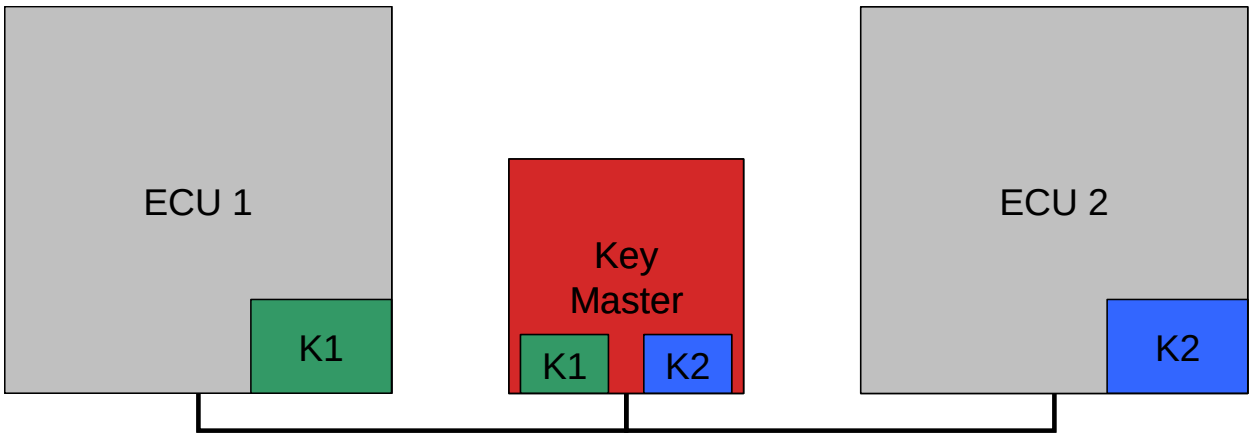

Abbildung 8: Evita Key Master Architektur

Die folgenden Szenarien werden abhängig von der verwendeten Hardware unterschieden:

(1) Gruppe aus **Full/Medium HSMs**, d.h. nur HSMs mit Unterstützung für asymmetrische Kryptographie. Hier können zwei Fälle unterschieden werden:

(a) Zwei Kommunikationspartner:
Über die Zertifikatstruktur können HSMs feststellen, dass ein Kommunikationspartner authentisch/vertrauenswürdig ist und anschließend über Public Key Kryptographie symmetrische Sitzungsschlüssel aushandeln. [30]

(b) Gruppenkommunikation:
Zur effizienten Schlüsselverteilung für Gruppenkommunikation wird in EVITA nur der symmetrische Fall mit Unterstützung durch einen Key Master betrachtet [30]. Dieser nimmt die Anfrage nach einer Gruppenkommunikation durch den Sender entgegen, überprüft die Richtlinienkonformität der Anfrage über das Policy Decision Module (PDM) und verteilt anschließend an alle Kommunikationspartner einen symmetrischen Sitzungsschlüssel, bevor der Erfolg der Anfrage an den Sender zurückgemeldet wird.

(2) Gruppe aus Full/Medium HSMs und insbesondere **mindestens einem Light HSM**, das nur symmetrische Kryptographie unterstützt. Auch hier können zwei Fälle unterschieden werden:

(a) Zwei Kommunikationspartner:
Ohne Unterstützung für asymmetrische Kryptographie ist eine vertrauenswürdige dritte Partei, der Key Master, nötig. In diesem Szenario benötigt mindestens das Light HSM einen gemeinsamen symmetrischen Schlüssel mit dem Key Master, der dann die Aushandlung eines Sitzungsschlüssels mit dem Medium/Full-Kommunikationspartner durchführen kann.

Anwendbarkeit

Die Schlüsselverteilung ist unter gewissen Umständen so verwendbar, es sollte aber vermutlich insbesondere unter Berücksichtigung existierender Sicherheitsprotokolle aus dem Internetkontext untersucht werden, inwiefern diese einfach wieder verwendbar sind. Durch die Wiederverwendung dieser gereiften Protokolle kann auf jahrelange Erfahrung mit diesen zurückgegriffen werden.

Vor- und Nachteile

(–) Redundanz von Key Mastern ggfs. Notwendig

(–) ggfs. Protokoll zur Synchronisation von Key Mastern notwendig

(+) Schlüsselmaterial für den Einsatz in Sicherheitsprotokollen aus der Internetwelt steht zur Verfügung

(+) Kommunikationsfähigkeit unmittelbar gegeben (Authentifizierung Key Master, Schlüsselaustausch)

(+) geringes zu verteilendes Wissen durch Dritte

(–) Schlüsselmaterial muss bei Austausch des Geräts im Key Master ebenfalls ausgetauscht weren

3.2.2 Platform Integrity Checks for Multi-Purpose ECUs

Multi-purpose ECUs fassen mehrere Funktionen, die zuvor auf unterschiedliche ECUs verteilt waren, in einer ECU zusammen.

Charakterisierung

Um Integrität, Authentizität und Vertrauenswürdigkeit der einzelnen Funktionen zu gewährleisten und sie gegeneinander abzuschotten, werden Virtualisierungstechniken eingesetzt, welche mit Trusted-Computing-Technologien (z.B. EVITA HSM oder TPM) kombiniert werden. Eine Multipurpose ECU besteht aus folgenden Komponenten:

- Hardware-basierter Vertrauensanker (z.B. EVITA HSM oder TPM)
- Hardwareplattform die Virtualisierung unterstützt (z.B. Intel Atom)
- Hypervisor
- Mehrere virtuelle Maschinen (VMs)

Die einzelnen VMs werden durch den Hypervisor kontrolliert, um sie möglichst vollständig voneinander zu isolieren. Techniken wie Virtual Machine Inspection ermöglichen die Erkennung von ungültigen Zuständen und erlauben das Zurücksetzen der VMs auf bekannte, sichere Zustände.

Mittels Integrity Stage Checks wird die Integrität der Plattformkonfiguration sichergestellt bzw. zumindest werden Manipulationen erkannt. Hierzu werden bekannte Techniken wie Sealing verwendet.

Ein Attestattionsprotokoll ermöglicht einer ECU die Vertrauenswürdigkeit der Plattformkonfiguration einer anderen ECU zu überprüfen.

Anwendbarkeit

Der vorgeschlagene Ansatz mittels Integrity Stage Checks basiert auf dem Einsatz von Multipurpose ECUs und Virtualisierung.

Das vorgeschlagene Attestationsprotokoll basiert auf dem Einsatz eines HSMs oder TPMs als Vertrauensanker. Im Rahmen von SEIS wird zumindest eine ECU mit solch einem hardwarebasierten Schutz ausgestattet sein. Für diese ECU wäre das Protokoll grundsätzlich denkbar. Zur Verifikation muss die prüfende ECU jedoch Public-Key-Operationen durchführen.

Vor- und Nachteile

(+) Virtualisierung ermöglicht das Zusammenführen von Funktionen auf eine ECU. Somit kann die Anzahl der ECUs reduziert werden.

(–) Einsatz von Virtualisierung benötigt relativ hohe Ressourcen, die ggf. nicht verfügbar sind.

(–) Attestationsprotokoll ggf. effizienter realisierbar.

Anpassungs- und Erweiterungsbedarf

Die Attestationsprotokolle sollten für ECUs, die nicht mit einem HSM oder TPM ausgestattet sind, um softwarebasierte Ansätze erweitert werden.

3.2.3 Secure Bootstrapping Protocol

EVITA trifft die Annahme, dass zur Initialisierung von Komponenten vertrauenswürdige Entities vorhanden sind wie ein Trusted Policy Server (TPS), HSM oder Schlüsselmaster. Policies sind in sicheren Speicherbereichen gespeichert und es wird über sichere Kanäle kommuniziert. Das Bootstrapping umfasst:

- Verteilung von Policies
- Schlüsselmanagement und Initialisierung
- Sichere Kanäle
- Zeitsynchronisierung
- Überprüfung der Plattformintegrität

3.2.4 Policy Management and Access Control Management

Charakterisierung

Policies spielen eine zentrale Rolle in der EVITA-Architektur. Die in EVITA betrachteten Security-Policies umfassen Zugriffskontrollrichtlinien (Access Control Policies) sowie Richtlinien zur Einbruchserkennung (Intrusion Detection Policies). Zugriffskontrollrichtlinien unterteilen sich in Filterrichtlinien (Filtering Policies) und Endsystem-Zugriffskontrollrichtlinien (Endpoint Access Control Policies). Filterrichtlinien werden in Gateways umgesetzt, die dazu wie eine Firewall agieren. Endsystem-Zugriffskontrollrichtlinien bestimmen anhand der Herkunft, ob Anwendungen bestimmte Nachrichten verarbeiten dürfen.

Policies werden in das Fahrzeug vor Auslieferung eingespeist. Das Hochladen kann offline erfolgen, eine Aktualisierung per Fernzugriff ist ebenfalls möglich.

Zur Verwaltung von Policies sind drei Protokolle vorgesehen:

- Policy Update Protocol – dieses Protokoll dient der Aktualisierung von Security Policies. Es kann regelmäßig oder bei Bedarf ausgeführt werden, z.B. wenn wichtige Sicherheitsaktualisierungen vorliegen. Das Protokoll sorgt für eine integritätsgesicherte Aktualisierung, wobei auch Zertifikate eingesetzt werden. Eine Verschlüsselung der Policies ist optional.
- Policy Configuration Protocol – dieses Protokoll umfasst Funktionalitäten für einen sicheren Aufstart (Secure Bootstrapping) oder zur Policy-Konfiguration während der Laufzeit.
- Policy Enforcement Protocol – dieses Protokoll umfasst Mechanismen für lokale Policy-Entscheidungen (Local Policy Decision) als auch für verteilte Entscheidungen (Distributed Policy Decision)

Üblicherweise konfigurieren sich die Sicherheitskomponenten anhand vorgegebener statischer Policies. Policies werden durch einen Policy Administration Point (PAP) verwaltet und können dort von Policy Decision Points (PDPs) angefordert werden, falls dort noch keine Policies bekannt sind. Zugriffskontrollrechte werden durch XACML bzw. ein EVITA-proprietäres Format beschrieben. XACML-Spezifikationen können durch Transformationen in das eingebettete effizientere EVITA-Format überführt werden. Policies können statisch vorkonfiguriert oder dynamisch konfiguriert sein. In jedem Fall sind sie aber nach Aufstart des Fahrzeugs konfiguriert.

Zur Policy-Konfiguration werden Authorization Tickets und Security Configuration Tickets eingesetzt. Erstere werden in PDPs verwendet, um dort über Autorisierungsanfragen von PEPs zu entscheiden; kann ein PDP mangels Wissen nicht autonom über eine solche Anfrage entscheiden, kann der die Anfrage an einen weiteren (ggf. spezialisierteren) PDP weiterleiten. Die Kommunikation erfolgt in diesem Fall durch das CCM geschützt. Security Configuration Tickets kommen zur Konfiguration von PEPs zum Einsatz. Solche Tickets können zur Laufzeit abgefragt oder offline eingesetzt werden, z.B. durch Ablage in einer Konfigurationsdatei. Da eine solche initiale Konfiguration nur einmal erfolgt, wird sie in der Regel auch nicht in einer lokalen Policy-Datenbank hinterlegt. Software-Installation ist ein Sonderfall des Offline-Einsatzes.

Anwendbarkeit

Die generelle Architektur mit PEPs und PDPs ist auch übertragbar auf den SEIS-Kontext. PDPs können je Domain existieren.

Das Konzept autonomer PEPs, die teilweise über integrierte PDMs verfügen und so selbständig Policy-Entscheidungen treffen können, ist vermutlich aus Skalierbarkeits- und Robustheitsgründen weitgehend einem System vorzuziehen, das für alle Entscheidungen ständig mit PDPs kommunizieren muss.

Vor- und Nachteile

(–) Einsatz zentraler Komponenten wie PDPs oder PAPs stellt ein Robustheitsproblem dar.

(–) Häufige Anfragen von PEPs zu PDPs erhöhen den Kommunikationsaufwand und die Latenz.

(+) Einsatz von XACML als Standard

(–) Komplexität von XACML

(–) EVITA-spezifische Konfigurationsbeschreibung vermutlich nicht einsetzbar, da zu spezifisch.

Anpassungs- und Erweiterungsbedarf

Anpassung auf das Zonenmodell ist notwendig.

3.2.5 Security Protocols Specific for Dedicated Applications

EVITA adressiert 4 Themen in Bezug auf Protokolle für bestimmte Applikationen:

- Sichere Firmware Aktualisierungsprotokolle
- Aktualisierungen und Schlüssel im HSM
- Ersetzen von Hardware und Sensoren
- Sicheres Geräte-Integrationsprotokoll

Diese werden im Folgenden erläutert.

Sichere FW-Aktualisierungsprotokolle

Evita stützt sich dabei auf die HIS-Flash-Lader Spezifikation (HIS – Herstellerinitiative Software) mit folgenden Elementen

- Neue Firmware Freigabe
- Prozedur für das Herunterladen
- Entfernte Verbindungsanfrage
- Erzeugen eines Sitzungsschlüsselpaares
- Schlüsselimport
- Flashlader – Laden der Flashdaten in das Stuergerät

Aktualisierungen und Schlüssel im HSM

Schlüssel, die innerhalb eines HSM gespeichert sind können an eine bestimmte Konfiguration der Plattform gebunden sein. Diese Konfiguration hängt inhärent von der installierten Firmware ab. Durch eine Aktualisierungsprozedur können diese Schlüssel ihre Gültigkeit abhängig von der Bindung verlieren. EVITA adressiert dafür folgende Lösungen:

Schlüsselaustausch:
Alle Schlüssel die an eine spezifische Plattform gebunden sind, werden durch die korrespondierende Zertifizierungsstelle ausgetauscht. Das erfordert eine Online-Infrastruktur und komplexe Protokolle außerhalb des Fahrzeugs und resultiert in nichtdeterministischen Aktualisierungszeiten.

Schlüsselaktualisierungen:
Die Tatsache, dass eine Aktualisierungsprozedur nur in einer autorisierten und authentisierten Weise angestoßen werden darf, garantiert, dass eine Folgeversion ähnlich zum Originalsystem ist, für die ein Schlüssel ausgegeben wurde. Bei einer größeren Umstrukturierung der Schlüsselstruktur müssen die neuen Schlüssel mit den alten signiert worden sein. Dadurch treten neue Sicherheitsrisiken auf, da ein Aktualisierer damit indirekt das gleiche Vertrauenslevel hat wie jede andere Zertifizierungsstelle. Daher ist dieser Ansatz eher theoretisch.

Schlüssel an frühe Bootphase binden:
Ein pragmatischer Weg, die Nutzbarkeit von konfigurationsgebundenen Schlüsseln nach einer Aktualisie-

rung zu erreichen, ist sie an eine frühe Konfigurationsphase zu binden, d. h. nachdem der Bootlader den Authentifizerungsprozess der Firmware beendet hat. Ein potentieller Nachteil ist, dass der Bootlader selbst während der Aktualisierung nicht verändert werden darf, was selten der Fall ist. Falls doch müssen die betroffenen Schlüssel geändert werden.

Zusätzliche ECR (ECU Configuration Registers)-Markierung:
Eine zusätzliche binäre ECR-Markierung, die den Zustand „sicher gebootet“ bezeichnet, kann eingeführt werden. Die Nutzung dieser Markierung hängt stark von der Sicherheit des Aktualisierungsprozesses ab. Darüber hinaus sollen Schlüssel nicht an ein bestimmtes Gerät gebunden sein, was die Idee der ECR Authentifizierungsmarkierung hinfällig macht

Zusammenfassend wird von der Annahme ausgegangen, dass der Bootlader in typischen Aktualisierungsszenarien nicht verändert wird und damit Autorisierungsmarkierungen von HSM-Schlüsseln nicht beeinträchtigt werden, welche im Booloader hinterlegt sind.

Ersetzen von Hardware und Sensoren

Annahme:
Ein Sensor s hat nur ein leichtes HSM installiert, d. h es bietet nur symmetrische Verschlüsselung. Ein Geheimnis - Device Identity Key (IDK-s, nachfolgend k-s genannt) ist bereits im HSM installiert. Der Sensor wird in einer offiziellen Werkstatt installiert, die entweder eine Online-Konnektivität zum Hersteller hat oder Teil vom Zertifizierungsprogramm des Herstellers ist, also ein gültiges Zertifikat besitzt.

Problem:
Der Schlüssel k muss sicher in den Schlüsselmaster der betreffenden Domäne des Sensors transportiert werden; jedoch k kann nicht aus dem HSM exportiert werden (d.h. er darf nicht ohne Transportverschlüsselung exportiert werden, jedoch gibt es keinen geteilten Schlüssel für Transportverschlüsselung).

Lösung:
Unterscheidung in eine Online- und Offline-Variante. Diese stützen sich auf unterschiedliche Vertrauensanker: Die Online-Methode erzeugt das Vertrauen über ein Hersteller-Backend. Hierbei wird der öffentliche Schlüssel des IDK an das Herstellerbackend übertragen und ein hiermit verschlüsselter, authentischer und zertifizierter Container mit den benötigten symmetrischen Schlüsseln wird an das Fahrzeug zrückgeliefert. Der entsprechende Key Master kann hierauf den Schlüssel importieren.

Bei der Offline-Lösung wird eine vom Hersteller freigegebene manipulationssichere Hardware benutzt. Der benötigte Schlüssel wird zusammen mit der Hardware (z.B. dem Sensor) mit dem öffentlichen Schlüssel der Werkstatt verschlüsselt ausgeliefert. Der zertifizierte Diagnosetester mit dem eigenen HSM kann den Schlüssel nun entschlüsseln und mit dem öffentlichen Schlüssel des Key Masters neu verschlüsselt an das Fahrzeug übertragen.

Sicheres Geräte-Integrationsprotokoll

Zum sicheren Anbinden und Integrieren von mobilen Geräten schlägt EVITA ein Protokoll vor, das einen authentischen und vertraulichen Kommunikationskanal zwischen mobilem Gerät und Fahrzeug herstellt. Dazu erfolgt gegenseitige Authentifizierung zwischen Fahrzeug und mobilem Gerät. Vor der Authentifizierung wird geprüft, ob die beteiligten Instanzen die erforderlichen Autorisierungen haben. Nach erfolgreicher Authentifizierung wird ein Sitzungsschlüssel zur vertraulichen Kommunikation eingerichtet.

3.2.6 Transportprotokoll

Da viele Anwendungen im Bordnetz nur Authentizität und Integrität erfordern, werden auf Protokollebene keine Verschlüsselungsmechanismen definiert.

Charakterisierung

Hauptdesignziele sind Effizienz, Flexibilität und Skalierbarkeit. Security Features sind nicht Bestandteil des Transportprotokolls und werden auf die Anwendungsebene verlagert. Security (MAC, Timestamps, etc.) ist also nur innerhalb der Payload vorgesehen.

Als Transportprotokoll wird CTP (Common Transport Protocol) vorgeschlagen; dies unterstützt auch Multicasting. Flow Control und Nachrichtenbestätigung (Acknowledgements) sind möglich, aber nur im Unicast-Betrieb.

Übertragen werden kann CTP über CAN, FlexRay und TCP/IP.

Anwendbarkeit

Eine Anwendbarkeit in SEIS ist kaum gegeben.

Security Features sollten innerhalb von SEIS auch schon auf den unteren ISO/OSI Schichten definiert und nicht nur applikativ betrachtet werden.

Vor- und Nachteile

(+) Fragmentierung wird unterstützt

(+) Multicastunterstützung

(+) Unterstützung für Acknowledgements

(+) Flexibel bzgl. der darunterliegenden Schichten (CAN, FlexRay,…)

(–) ineffizient über IP

(–) keine integrierten Security Features

4. KRYPTO-MODUL UND HARDWARE SECURITY

Dieses Kapitel betrachtet das CSM (Cryptographic Services Module) und teilweise das SSM (Secure Storage Module, siehe hierzu auch Kapitel 7: Key-Management der SEIS-Security-Architektur und die Möglichkeit und Notwendigkeit der Verankerung von Security-Funktionalität in (vertrauenswürdiger) gesondert geschützter Hardware.

Das CSM stellt lokal eine einheitliche Schnittstelle für Kryptofunktionen, -services und –primitive zur Verfügung, die von der konkreten Implementierung abstrahiert. Die Implementierung selbst kann dann in Software oder Hardware, lokal oder remote in verschiedener Kombination und auch in Mischformen erfolgen. Die Schnittstelle soll jedoch sicherstellen, dass die entsprechenden Grundsätze an Schlüsselmanagement, Autorisierung, Nutzungsrichtlinien und Schutz der Implementierung unterstützt und wo möglich durchgesetzt werden. Im Folgenden bezieht sich die Bezeichnung Krypto-Modul oder CSM im engeren Sinne auf diese lokale Schnittstelle, unabhängig von der konkreten Implementierung.

Das SSM wird als Schlüsselspeicher im Wesentlichen durch die in Kapitel 7 eingeführte Schnittstelle angesprochen. Da für die Ausführung kryptographischer Funktionen jedoch direkter Zugriff auf geheimes Schlüsselmaterial erforderlich ist, arbeitet es eng mit dem CSM zusammen. Im Fall von in Hardware implementierten Algorithmen oder allgemeinerer Hardwareunterstützung für Kryptoalgorithmen ist ein spezieller Schutz der Hardware und des Schlüsselzugriffs sinnvoll und naheliegend, so dass eine gemeinsame Implementierung in einem Hardware Security Modul (HSM) Vorteile bietet. Möglichkeiten, Aufgaben und Anforderungen für ein solches Hardware Security Modul werden ebenfalls betrachtet.

4.1 MOTIVATION

Viele der in den vorangegangenen Kapiteln angesprochenen und verwendeten Sicherheitsmechanismen verwenden wiederum zur Erreichung ihrer Sicherheitsziele kryptographische Primitive, die ihrerseits auf der Verwendung kryptographischer Schlüssel oder anderer sensibler Daten beruhen, deren Integrität, Authentizität und/oder Geheimhaltung Grundannahmen für die Sicherheit des Gesamtsystems sind. Des Weiteren beruht die Korrektheit und Vertrauenswürdigkeit jedwelcher Mechanismen auf der Vertrauenswürdigkeit der darunterliegenden Hardware und/oder Software, deren Schnittstellen, Funktionen oder Ausführungseinheiten verwendet werden.

Die zuverlässige, vertrauenswürdige Bereitstellung der benötigten Ressourcen und Primitive ist damit wesentliche Grundlage für die Sicherheit und Wirksamkeit aller vorgestellten Verfahren. Abhängig vom verwendeten Angreifermodell sind verschiedene Level der Absicherung von Verfahren und Ausführungseinheiten (bspw. auch gegen Seitenkanalangriffe) notwendig.

Security-Mechanismen werden von verschiedenen Applikationen und auf verschiedenen Ebenene des Kommunikationsstacks verwendet. Da speziell bei Anwendungen Dritter nicht immer und überall die Einhaltung aller Sicherheits- und Kodierrichtlinien überprüft werden kann, ist eine Kapselung der kritischsten und wichtigsten Funktionen und speziell ein strikt reglementierter Umgang mit geheimem (Schlüssel-) Material ein wichtiger Schritt zur Absicherung des Gesamtsystems.

Das folgende Kapitel widmet sich der Kapselung von Krypto-Funktionalität im CSM und der Realisierung, Unterstützung und Verankerung von Security-Funktionalität und Vertrauenswürdigkeit in Hardware.

4.2 ANGREIFERBETRACHTUNG

Grundlage jeder Sicherheitsbetrachtung ist ein Angreifermodell, das die Fähigkeiten und Möglichkeiten zur Einflussnahme durch den Angreifer und ggf. seine Intention beschreibt. Gegen dieses Modell kann ein System gehärtet bzw. seine Sicherheit oder Resistenz evaluiert werden. Im hier vorliegenden Fall ist die allgemeine Definition eines Angreifermodells schwierig, da eine Vielzahl verschiedener Anwendungsfälle und Subsysteme betrachtet werden, die in Kritikalität, Angreifbarkeit und Anreiz für einen potentiellen Angreifer stark voneinander abweichen. Um trotzdem eine Sicherheitsbetrachtung und einen Vergleich verschiedener Ansätze zur Absicherung der verwendeten Primitive anstellen zu können, soll hier eine

allgemeine Betrachtung erfolgen, was die Angreifer auf eingebettete, speziell automobile Systeme ausmacht und was sie ggf. grundsätzlich von Angreifern auf PC- oder Web-basierte Systeme unterscheidet.

Eine wesentliche Eigenschaft des eingebetteten Automotive-Gesamtsystems und vieler Teilsysteme ist die Safety-kritische Funktionalität. So können Manipulationen und Fehlfunktionen einerseits potentiell Gefahr für Leib und Leben von Insassen und Unbeteiligten bedeuten, was die Wichtigkeit des Schutzes unterstreicht. Andererseits müssen alle Reaktionen auf erkannte Angriffe oder Fehlfunktionen immer berücksichtigen, dass Fail-Safe-Modes eingenommen und angegriffene Systeme nicht unbedingt einfach abgeschaltet werden können.

Allgemein gibt es einige wesentliche Unterschiede eingebetteter Systeme im Vergleich zu klassischen PC-Systemen. Einer besteht darin, dass sie meist eine klar definierte Aufgabe erfüllen, ohne von Eigentümer/Benutzer frei konfigurierbar oder erweiterbar zu sein. Sie haben damit eine deutlich statischere Struktur und oft eine feste Hard- und Softwarekonfiguration. Während dies die Absicherung eher erleichtert, erschweren andere Eigenschaften die Sicherstellung von Security. So muss etwa bei Fahrzeugsystemen davon ausgegangen werden, dass ein potentieller Angreifer vollen physikalischen Zugriff auf die angegriffenen Systeme hat. Dies kann entweder auf ein auf einer öffentlichen Straße parkendes Fahrzeug geschehen, oder sogar in einer geschützten Umgebung, etwa wenn ein Angriff in Kooperation mit einer Werkstatt oder dem Eigentümer selbst erfolgt. Speziell in letzterem Fall ist es zumindest wichtig, eine Manipulation später nachweisen zu können, um etwa Produkthaftungsfragen eindeutig klären zu können.

Je nach Anwendungsfall muss also von einem Angreifer ausgegangen werden, der über einen längeren Zeitraum vollen physikalischen Zugriff auf das System und seine Schnittstellen hat, während das System selbst zuallererst die Safety sicherstellen muss und die Absicherung der Security hiernach lediglich zweite Priorität haben darf. Hierauf wird bei der Auswahl und Realisierung von Sicherheitsmaßnahmen und Hardwareverankerung zu achten sein.

4.3 KRYPTO-SERVICES

Viele Applikationen, aber auch viele Dienste (unter anderem die meisten der betrachteten Security-Maßnahmen zur Kommunikationshärtung) benötigen kryptographische Verfahren wie Hashing, Signaturerstellung und -verifikation, HMACs oder Ver-/Entschlüsselung mit unterschiedlichen Verfahren. Einige Gründe legen nahe, diese Verfahren den Anwendungen und Diensten als Service zur Verfügung zu stellen, unter anderem sind hier zu nennen:

- Einheitliche Schnittstellezur Anwendung
- Abstraktion von Implementierungen, und zwar sowohl von der Implementierungsart (in Software, in Hardware, oder eine Kombination), als auch vom Implementierungsort (lokal auf dem aufrufenden Steuergerät, remote/zentral auf einem Steuergerät als Service oder sogar als verteilt realisierter Dienst).
- Schlüsselmanagement und Schutz von geheimem Schlüsselmaterial. Die Schnittstelle stellt hierzu keinen direkten Zugriff auf die Schlüssel zur Verfügung, sondern ausschliesslich auf Funktionen, die die Schlüssel verwenden.
- Performanz (ggf. Einsatz von HW-Unterstützung wo nötig)
- Spezieller Implementierungsfokus auf Sicherheit: Speziell Seitenkanalangriffe gewinnen immer mehr an Relevanz und stellen inzwischen ein ernstzunehmendes Risiko for die Sicherheit von Systemen dar. Die gekapselte Implementierung von Security-kritischen Funktionen und Services bietet hier die Möglichkeit, in dieser abgeschlossenen Domäne mit speziellem Expertenwissen und security-zentrierten Richtlinien und Prozessen bei der Implementierung einen Sicherheitslevel zu erreichen, der angewendet auf die Gesamtimplementierung nicht oder nur zu sehr hohen Kosten erreichbar wäre.

Dieser Ansatz wird auch in SEIS verfolgt. Auf die konkret angebotenen Services und ihre Eigenschaften wird in Abschnitt 4.8 genauer eingegangen.

Aufgabe des Krypto-Moduls ist es eine einheitliche und transparente Schnittstelle zu den Krypto-Services zu bieten, welche die Implementierungsdetails verbirgt. Die angebotenen Services können über verschiedene Wege realisiert werden, sei es rein in Software oder mit Hardware-Unterstützung, lokal oder entfernt auf einer anderen ECU.

4.3.1 Lokale Servicebereitstellung

Die oben beschriebenen Dienste kann das Krypto-Modul je nach Leistungsstärke direkt lokal auf dem jeweiligen Steuergerät bereitstellen. Dies kann über reine Software- oder Hardware-basierte Lösungen aber auch über hybride Implementierungen erfolgen, bei denen die Hardware nur bestimmte Funktionen übernimmt. Die vom Krypto-Modul angebotene Schnittstelle verbirgt diese Implementierungsdetails.

Hardware-basierte Technologien können einerseits als kryptographische Beschleuniger für nicht ausreichend leistungsstarke CPUs oder Mikrocontroller wirken, andererseits bieten sie je nach verwendeter Technologie zusätzliche Möglichkeiten, die Software-basierten Ansätze nicht bieten, wie etwa manipulations-resistente Berechnungen und Speicher. Abschnitt 4.6 behandelt die unterschiedlichen Hardware-Lösungen ausführlicher.

4.3.2 Entfernte Servicebereitstellung

Die Servicebereitstellung kann neben der lokalen Ausführung auch über andere Steuergeräte realisiert werden, die leistungsstärker sind oder mit aufgrund spezieller Hardware-Austattung erweiterte Fähigkeiten besitzen. In diesem Fall ergibt sich eine klassische Client-Server-Beziehung, bei der das Krypto-Modul die Service-Anfragen transparent an eine entfernte (Server-)ECU weiterleitet und die empfangenen Resultate in das lokale System wieder zurückspielt.

Die Funktionalität der im Abschnitt „7.4.2 Zentrale Schlüsselspeicherung“ eingeführten KD-ECU kann z.B. die Server-Rolle einnehmen und um die Bereitstellung von kryptographischen Services erweitert werden. Die KD-ECU kann neben der Schlüsselverwaltung auch Funktionen wie das Erstellen und Verifizieren von Signaturen, kryptographische Hashes, Zufallszahlenerzeugung, sowie asymmetrische Ver- und Entschlüsselungsdienste anbieten.

Diese Dienste können insbesondere für leistungsschwache ECU von Nutzen sein, die keine Möglichkeit haben, etwa asymmetrische kryptographische Verfahren anzuwenden oder kritische Daten sicher zu speichern. Meist muss hier analog zu der Schlüsselverteilung der Kommunikationskanal abgesichert werden. Je nach Anwendung sind Vertraulichkeit, Integrität und Authentizität auf dem Kanal sicherzustellen. Hier könnten auch rein symmetrische Verfahren (z.B. mittels Pre-Shared-Key) für die Kommunikationsabsicherung zum Einsatz kommen, die weniger Voraussetzung an die ECU stellen (vgl. im EVITA-Ansatz ein HSM-small anstatt eines HSM-medium für die asymmetrische Verschlüsselung).

Ein weiterer interessanter Punkt ist die zusätzliche Sicherheit von besonders zu schützenden Schlüsseln. Falls es Schlüssel gibt, welche die KD-ECU oder sogar deren sicheren Speicher nicht verlassen dürfen und dies auch über die Policies so festgelegt ist, können Operationen mit den Schlüsseln direkt auf der KD-ECU oder innerhalb des (dortigen) HSMs durchgeführt werden und somit eine Übertragung auf potentiell unsicherere ECUs und über unsichere Kanäle vermieden werden.

Bei der Verwendung der entfernten Servicebereitstellung gibt es anwendungsspezifisch einige Fragestellungen, die im Einzelfall zu untersuchen sind:

- Welche Latenzen (durch Kommunikation und Berechnung) entstehen durch die Verlagerung der Berechnungen auf die Server-ECU?
- Welche Rechenlast entsteht auf der Server-ECU? Wie sind die Anfragen verteilt? Sind evtl. „Lastspitzen“ vorhanden, z.B. beim Start der Fahrzeugs?
- Welcher Kommunikations-Overhead und welche zusätzliche Buslast entstehen durch die zusätzliche Kommunikation zur Server-ECU?
- Welche sicheren Kanäle sind nötig zwischen Client- und der Server-ECU?
- Wie wird eine sichere und zuverlässige Authentifizierung und Autorisierung beim Zugriff auf die Krypto-Services sichergestellt?

4.4 UMGANG MIT SCHUTZWÜRDIGEN DATEN

Der Umgang mit schutzwürdigen Daten ist ein zentrales Thema beim Entwurf einer Sicherheitsarchitektur. Es muss genau abgewägt werden, welche Module Zugang zu diesen Daten benötigen und wie dieser zu gestalten ist. Um die Möglichkeit von Manipulationen und somit Missbrauch einzuschränken, sollte der Zugriff auf kritische Daten restrikiv gehandhabt werden und nur wenige überprüfbar vertrauenswürdige und autorisierte Module sollten Zugang haben.

Daher wird aus Sicherheitssicht das System / die Middleware in SEIS als in zwei Bereiche aufgeteilt betrachtet: die sog. „Protected Area“ und der Rest. Module innerhalb der Protected Area haben direkten Zugriff auf Geheimnisse, also schutzwürdige Daten, Module außerhalb haben nur Rechte an Operationen auf diesen Daten, die innerhalb der Protected Area ausgeführt werden.

Die Protected Area besteht aus zwei Modulen: dem Key Management als Geheimnisverwahrer und dem Krypto-Modul als ausführende Entität. Die Vertrauenswürdigkeit der Protected Area wird mittels einer Vertrauenskette, beschrieben in Abschnitt 4.5, nachgewiesen.

Die zur Ausführung kryptographischer Operationen nötigen Schlüssel sind vertraulich zu behandeln und sollten im Regelfall nur in sehr vertrauenswürdigen Umgebungen verwendet werden. Um die Verbreitung zu minimieren werden alle Funktionen, die direkten Zugriff auf Schlüssel benötigen, im Krypto-Modul gekapselt. Somit greift grundsätzlich nur das Krypto-Modul direkt auf Schlüsseldaten zu, wohingegen andere Module und Applikationen nur mit Handles auf Schlüssel arbeiten. Abbildung 9 zeigt schematisch den Umgang mit Schlüsseln innerhalb und außerhalb der Protected Area bei Inanspruchnahme eines Krypto-Service von Applikationsseite.

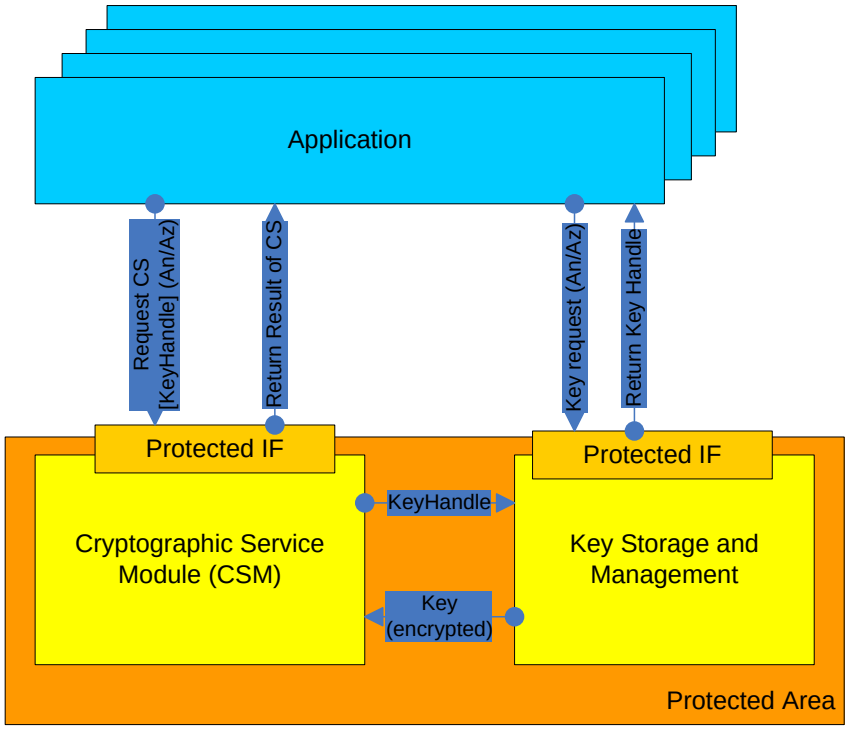

Abbildung 9: Schematische Darstellung des Umgangs mit Schlüsseln zwischen SSM, CSM und Applikationen

Neben der vertrauenswürdigen Behandlung von Schlüsseldaten, stellt die Speicherung kritischer Daten / Geheimnisse eine weitere wichtige Aufgabe des Krypto-Moduls dar. Daher bietet das Krypto-Modul Services, welche kritische Daten je nach Bedarf integritätsgeschützt, authentisch und/oder vertraulich verwahrt. Dies kann in Form von verschlüsselten oder signierten Datenobjekten geschehen.

Abbildung 10 veranschaulicht den Umgang mit Schlüsseldaten bei einer Verschlüsselungsoperation.

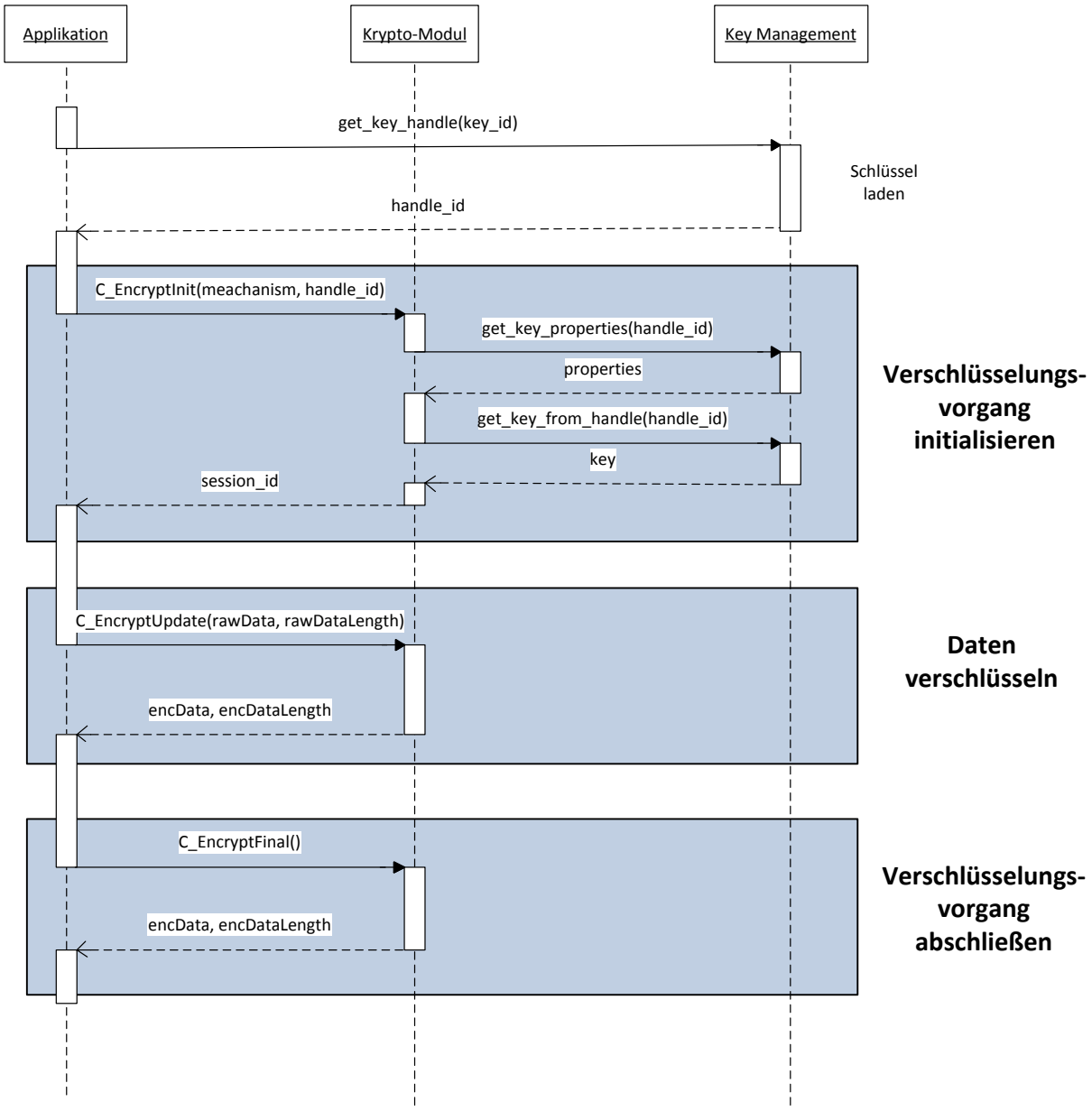

Abbildung 10: Umgang mit Schlüsseldaten

Abbildung 10 zeigt zunächst, wie eine Anwendung beim Key Management unter Vorlage der entsprechenden Autorisierung einen Handle zum gewünschten kryptographischen Schlüssel anfordert. Als nächstes wird durch den Aufruf von „C_EncryptInit()“ mit dem erhaltenen Handle und der Angabe des zu verwendenden kryptographischen Verfahrens die Verschlüsselung der Daten beim Krypto-Modul initiiert. Das Krypto-Modul fordert intern die realen Schlüsseldaten vom Key Management an, das ausgeliefert wird, falls das Krypto-Modul als vertrauenswürdig erachtet wird.

Es obliegt der Verantwortung des Krypto-Moduls, die vom Key Management erhaltenen Schlüsseldaten ihrer festgelegten Bestimmung nach zu verwenden. D.h. wenn ein Schlüssel z.B. die Eigenschaft hat, dass man mit ihm nur Signieren darf, muss das Krypto-Modul sämtliche anderen Operationen auf Basis dieses Schlüssels ablehnen. Das Krypto-Modul dient somit als Policy Enforcement Point (PEP) für die bestimmungsgemäße Schlüsselverwendung. Deshalb werden ebenfalls die Eigenschaften des Schlüs-

sels angefordert, welche die erlaubten Verwendungsmöglichkeiten des Schlüssels spezifizieren und das Krypto-Modul die geforderte Operation auf ihre Autorisierung überprüfen kann. Des Weiteren ist das Krypto-Modul in seiner Funktion als PEP und Schlüsselverwender ein Trusted Device und ist entsprechend der jeweiligen Sicherheitsanforderungen abzusichern.

Der Verschlüsselungsvorgang wird anhand einer Session identifiziert, die der aufrufenden Anwendung in Form eines Handles mitgeteilt wird.

Das eigentliche Verschlüsseln der Daten wird von der Anwendung durch (falls nötig wiederholte) Aufrufe von „C_EncryptUpdate()“ durchgeführt. Mittels „C_EncryptFinal()“ wird der Verschlüsselungsvorgang abgeschlossen und der Session Handle entwertet.

4.5 VERTRAUENSKETTE

Da das Krypto-Modul direkt Operationen anhand schutzwürdiger Daten (z.B. kryptographische Schlüssel) durchführt, muss dessen Vertrauenswürdigkeit nachweisbar gestaltet werden. Dies kann durch den Einsatz von Komponenten erreicht werden, die tamper-resistant oder tamper-evident sind. Tamper-resistant bedeutet, dass man diese Komponente gar nicht oder nur mit hohem Aufwand manipulieren kann. Dies sind in erster Linie sichere Hardware-Module, deren Datenspeicher und Logik speziell gegen Software- und Hardware-Angriffe gehärtet sind. Näheres hierzu in Abschnitt 4.6. Tamper-evident ist eine schwächere Eigenschaft von Schutzmechanismen, die Manipulationen zwar nicht verhindern können, deren Durchführung jedoch im Nachhinein erkennbar und nachweisbar machen.

Die Vertrauenswürdigkeit von Teilen oder einer ganzen ECU kann durch tamper-evident Software nachweisbar realisiert werden. Hierbei wird aber ein Vertrauensanker („Root of Trust” (RoT)) benötigt, der eine Verifizierung der Software ermöglicht, indem er Charakteristika eines vertrauenswürdigen Zustandes sicher zum Vergleich bereitstellt.

Diese RoT kann z.B. in lokaler Form oder über entfernte tamper-resistant Module realisiert werden. Ebenfalls denkbar ist die Vernetzung von RoTs und das Bilden eines Trust-Networks.

Aufbauend auf einem verfügbaren RoT gibt es Software-basierte Verfahren, welche den Zustand der Software anhand von in der RoT sicher hinterlegten Daten verifizieren können:

- Secure Boot Mechanismen (rein Software-basiert oder Hardware-unterstüzt)
- Trusted Operating System (Isolierung von Applikationen oder Middleware-Modulen durch Virtualisierungstechniken)
- Ausführen ausschließlich zertifizierter Software (signiert vom OEM)
- Remote-Attestation

Auf Techniken aus dem Bereich Attestation wird genauer in [14], Kapitel 3.5 eingegangen. Generell ist eine höhere Sicherheit durch hardwarebasierte Attestation zu erreichen, wie sie bspw. von der Trusted Computing Group für TPM-basierte Systeme vorgeschlagen wird [27]. Allerdings setzt dies auf jedem zu attestierenden System einen vertrauenswürdigen Hardware-Baustein, etwa ein TPM voraus, was im Vergleich zu softwarebasierter Attestation üblicherweise mit zusätzlichen Kosten verbunden ist.

4.6 HARDWARE SECURITY SUPPORT

Die Vertrauenswürdigkeit des Gesamtsystems und die Korrektheit und Sicherheit des CSM basiert auf der Korrektheit, Zuverlässigkeit und Manipulationssicherheit der verwendeten kritischen Funktionalität, Als grundlegend für ein sicheres und vertrauenswürdiges System können drei Eigenschaften / Funktionen hervorgehoben werden:

- Sichere Speicherung von Schlüsseln
- Sichere Krypto-Services
- Vertrauensanker (RoT) für Plattformintegrität und ggf. Attestierung

Idealerweise erfolgt die entsprechende Realisierung in Hardware, bspw. in einem speziellen Hardwarebaustein, der im Folgenden als Hardware Security Module (HSM) bezeichnet werden soll. Der Einsatz spezieller Sicherheitshardware stellt jedoch einen wesentlichen Zusatzaufwand dar, so dass er vor allem aus Kostengründen gründlich abgewogen werden muss. Allerdings hat der Einsatz sicherer Hardware einige grundlegende Vorteile, die prizipiell in Software nicht oder nur mit sehr großem Aufwand realisiert werden können:

- Reine Softwaresysteme können die zugrundeliegende Hardwarebasis nur schwerlich selbst überprüfen (selbst PUF müssen in der Hardware geschützt und sicher ausgelesen werden).
- Jede (Vertrauens-)Kette ist nur so stark und nützlich wie jedes einzelne ihrer Glieder und insbesondere ihre Verankerung.
- Softwaresysteme können sich selbst nicht gegen Hardwareangriffe schützen.
- Geheime (Schlüssel-)Daten müssen sowohl im stromlosen Zustand als auch während der Verarbeitung gegen Manipulation und Auslesen gesichert sein (gegen andere Tasks, Komponenten, Externe). Dies schließt je nach Angreiferprofil insbesondere auch den Schutz gegen physikalische und Seitenkanalangriffe ein.
- Standard-Speicher können relativ leicht ausgelesen werden.

Die Kapselung sicherheitskritischer Funktionalität in einen eigenen, geschützen Hardwarebereich ermöglicht es, im gesamten Systementwurf ein besonderes Augenmerk auf die Sicherheit zu legen und das System mit einer möglichst einfachen, klaren, geschützen Schnittstelle zu versehen. Neben dem besseren Schutz degen physikalische Angriffe entfallen damit auch potentielle Angriffe durch Einflüsse nebenläufiger, eigentlich nicht security-relevanter Prozesse auf derselben Ausführungseinheit.

Einen weiteren Hardwareaspekt stellt die Integration von spezieller Hardwareunterstützung, -beschleunigung oder Komplettimplementierung von Security-Algorithmen dar. Viele kryptographische Verfahren verwenden spezielle mathematische Strukturen und Verfahren (etwa Langzahl- oder Modulararithmetik, bspw. für RSA und ECC), die einerseits für Standard-CPUs in der Berechnung sehr aufwendig sind, andererseits sehr effizient in Hardware implementiert werden können. Andere Verfahren wie bspw. Hashing und Blockchiffren basieren auf Grundfunktionen wie XOR (mit geheimen Schlüsseln), Permutationen (PBOX) und Ersetzung von Teilblöcken (Substitution - SBOX), ebenfalls schnell und effizient in Hardware umzusetzen sind. Je nach Anforderungen an Durchsatz und Latenz, aber auch abhängig von der Auslastung der Host-CPU ist Hardwareunterstützung unabhängig von der Sicherheitsbetrachtung zur Erreichung der Performanzanforderungen notwendig. Ein Beispiel hier ist die Unterstützung für Car2X-Kommunikation (vgl. hierzu etwa EVITA-full [32]).

4.7 MAPPING AUF VERFÜGBARE ANSÄTZE

In SEIS wurden keine eigenen HSM-Ansätze entwickelt, jedoch kommen für den Einsatz in den in SEIS betrachteten Systemen einige bekannte Entwürfe für HSMs in Frage, die je nach Einsatzzweck unterschiedlich geeignet sind. Einen vergleichenden Überblick über die Eigenschaften und Features gibt bereits Kapitel 3.

4.7.1 Trusted Platform Module (TPM)

Das von der Trusted Computing Group (TCG) spezifizierte Trusted Platform Module (TPM) [27] ist ein als Vertrauensanker für den PC-Bereich entworfenes Modul mit umfangreicher Funktionalität. Es bietet Unterstützung für Trusted Boot-Prozesse, sichere Speicherung von Schlüsseln, Beglaubigung (Attestation) der Konfiguration der eigenen Plattform (auf der es verbaut ist) und Binden von Schlüsseln an Plattformen und Konfigurationen. Obwohl es vom Entwurf her ausschließlich gegen Softwareangriffe Schutz bieten soll, ist in der Spezifikation eine gewisse Manipulationsresistenz gefordert, was gerade im eingebetteten Bereich wichtig ist. Vom Funktionsumfang bietet es alle Möglichkeiten, zur Absicherung von ECUs bis zu bspw. zentralem Schlüsselspeicher (KD-ECU) eingesetzt zu werden. Allerdings erfordert die Nutzung umfangreiche Unterstützung des Betriebssystems (z.B. für Trusted Boot). Außerdem wird davon ausgegangen, dass für umfangreiche und speziell zeitkritische Kryptooperationen eine leistungsfähige Host-CPU zur Verfügung steht. Die eigenen Kryptofunktionen sind auf das absolut notwendige beschränkt (bspw. erfordert die Spezifikation keine Unterstützung für symmetrische Kryptographie) und nicht auf Performanz optimiert. Hier wäre im Automotive-Umfeld ggf. eine Integration von

Kryptobeschleunigern sinnvoll. Des Weiteren werden nur manche Kryptoverfahren unterstützt (bspw. RSA, nicht jedoch ECC). Hier ist in der neuen Version 2.0 eine Erweiterung geplant. Außerdem sind die meisten verfügbaren TPM-Bausteine nicht für den Einsatz in Automotive-Systemen qualifiziert. Zusammenfassend sind in Form von TPM-Bausteinen Hardwareanker für vertrauenswürdige Plattformen vorhanden, jedoch ist für den Automotive-Einsatz eine spezifischere Hardwareunterstützung wünschenswert. Es ist jedoch anzuraten, die weitere Entwicklung der Spezifikation und Standardisierung in der TCG in Hinblick auf die Automotive-Domäne zu beobachten. Zu einigen Domänen gibt es bereits spezielle Spezifikationen (vgl. etwa Mobile Trusted Module [26]) und Arbeitsgruppen (bspw. zu eingebetteten Systemen, siehe [25]).

4.7.2 Secure Hardware Extension SHE.

Die Secure Hardware Extension (SHE) wurde für den Automotive-Bereich in der Herstellerinitiative Software (HIS) [13] spezifiziert. Sie unterstützt symmetrische Kryptographie (AES) in Hardware zur Verschlüsselung und zum Integritätsschutz (MAC) und ist für einfache Systeme wie etwa Sensoren einsetzbar zur Kommunikationsabsicherung. Für sicherheitskritische Systeme mit umfangreicherer Securityfunktion ist die Funktionalität jedoch nicht ausreichend.

4.7.3 EVITA-HSMs

Die von EVITA spezifizierten und prototypisch implementierten HSMs sind speziell für den Automotive-Einsatz entworfen und auf die Bedürfnisse abgestimmt. Es existieren drei verschiedene Varianten, die für verschiedene Einsatzszenarien und Steuergeräte optimiert sind, und die im Gesamtkonzept kombiniert und sich gegenseitig ergänzend eingesetzt werden. Eine detaillierte Darstellung findet sich in den EVITA-Ergebnisdokumenten [30] [32].

- EVITA small unterstützt wie SHE symmetrische Verschlüsselungs und Integritätsschutzmechanismen und ist als Basis-Absicherung für den Einsatz in Sensoren und einfachen Steuergeräten konzipiert. Es ist speziell im Fall einer zentralen Schlüsselspeicherung als Security-Unterstützung in einfachen Teilsystemen im SEIS-Kontext denkbar.
- EVITA medium stellt umfangreiche Security-Funktionalität in Form von symmetrischer (HW-beschleunigt) und asymmetrischer Kryptographie, Hashing und Integritätsschutz zur Verfügung und ist grundsätzlich als Vertrauensanker auch für zentrale Security-Systeme wie einen zentralen Schlüsselserver (KD-ECU) einsetzbar. Auch existieren erste Implementierungen für den Automotive-Bereich [5]. Es kommt grundsätzlich für alle in SEIS angesprochenen Anwendungsfälle und Teilsysteme in Frage, wenn es entsprechend von der Plattform unterstützt wird um bspw. Trusted Boot und Attestierung zu unterstützen.
- EVITA full erweitert das medium-Modul vor allem um HW-Beschleunigung für asymmetrische Kryptographie. Es kommt damit vor allem für die Unterstützung von externer Kommunikation zwischen Fahrzeugen (C2C/V2V bzw. C2X/V2X) in Frage. Da dies im Rahmen von SEIS nicht betrachtet wurde, ist das EVITA full-Modul aus momentaner Sicht für die SEIS-Anwendungsfälle nicht notwendig.

4.8 Anwendungsspezifisches Security API

Im Folgenden werden Schnittstellen-Funktionen definiert, welche die Nutzung des Krypto-Moduls ermöglichen. In diesem Kontext gibt es bereits den Standard PKCS#11 [24], welcher eine API zum Zugang und Nutzen von kryptographischen Token (z.B. Smartcards) definiert. Die hier vorgestellte API lehnt sich an diesem etablierten Standard an, ist jedoch aufgrund folgender Gegebenheiten eingeschränkt:

- Es wird kein Session-Management für den Zugang zum Krypto-Modul verwendet, da es ein Modul innerhalb des SecurityFrameworks ist und keine externe Entität darauf zugreift. Es wird jedoch eine kleine Abwandlung von Sessions verwendet, um sequenzielle Funktionsaufrufe (z.B. Verschlüsseln oder Hashing von Daten) semantisch zu verknüpfen.
- Es ist keine explizite Authentifizierung nötig, da die Funktionalität des Krypto-Moduls von ECU-internen Modulen aufgerufen und nicht extern angeboten wird. Autorisierungen (gebunden an die

verwendeten Schlüssel) werden jedoch implizit durch Überprüfungen beim Policy Descision Management durchgeführt.

- Es werden keine Funktionen zum Schlüssel Management bereitgestellt, da diese Aufgabe bereits das Key Management Modul übernimmt.

Die API bietet Funktionen von PKCS#11 aus den Bereichen:

- Encryption
- Decryption
- Message Digesting
- Signing and MACing
- Functions for verifying signatures and MACs
- Dual-Function Cryptographic Functions
- Random Number Generation

Zusätzlich bietet das Krypto-Modul weitere Funktionen, welche das sichere Speichern und Laden von schutzwürdigen Daten erlauben sowie spezielle Fähigkeiten verbauter Hardware-Module zugänglich machen.

Nachfolgend werden exemplarisch Funktionen vorgestellt, welche die Verschlüsselung von Daten ermöglichen.

4.8.1 Funktion: C_EncryptInit()

- Funktionsbeschreibung: Mit dieser Methode initiiert man einen Verschlüsselungvorgang und legt die für die Verschlüsselung nötigen Parameter fest.
- Input/Output-Format:

Parameter	Type	Direction	Description
EncryptionMechanism_id	Mechanism_ID	In	Identität des zu verwendenden kryptographischen Verfahrens.
EncryptionKey_id	Handle_ID	In	Handle auf den zu verwendenden kryptographischen Schlüssel.
EncryptionSession_id	Session_ID	out	Der erzeugte Session-Handle für den Verschlüsselungsvorgang.
	ReturnCode	return	„CSM_EncryptionInit_OK“: Das Initialisieren des Verschlüsselungsvorgangs war erfolgreich. „Key_not_found“: Es konnte kein Schlüssel gefunden werden, der mit dem Handle assoziiert ist. „Key_incompatible“: Der Schlüssel ist inkompatibel mit dem angegebenen Verschlüsselungsmechanismus. „Key_encryption_unauthorized“: Der Schlüssel ist nicht für Verschlüsselungsvorgänge vorgesehen.

4.8.2 Funktion: C_EncryptUpdate()

- Funktionsbeschreibung: Mit dieser Funktion werden Daten verschlüsselt. Dieser Funktion muss ein Aufruf von „C_EncryptInit()“ zur Erzeugung eines Session-Handles vorangegangen sein.
- Input/Output-Format:

Parameter	Type	Direction	Description
EncryptionSession_id	Session_ID	in	Der Session-Handle für den Verschlüsselungsvorgang.
Raw_Data	BYTE-Array_PTR	In	Zeiger auf den Array der zu verschlüsselnden Daten.
Raw_Data_Length	Int	in	Anzahl der zu verschlüsselnden Bytes in Raw_Data.
Enc_Data	BYTE-Array_PTR	In/out	Zeiger auf einen Array in dem die verschlüsselten Daten gespeichert werden.
Enc_Data_Length	Int_PTR	In/out	In: Größe des Arrays auf den Enc_Data zeigt. Out: Anzahl der verschlüsselten Bytes, die in Enc_Data gespeichert sind.
	ReturnCode	return	„CSM_Encryption_OK“: Das Verschlüsseln war erfolgreich. „CSM_Invalid_Session“: Es wurde eine ungültige Verschlüsselsungs-Session angegeben. „CSM_Enc_Data_Too_Small“: Der Array in Enc_Data ist zu klein.

4.8.3 Funktion: C_EncryptFinal()

- Funktionsbeschreibung: Mit dieser Funktion wird ein Verschlüsselungsvorgang abgeschlossen. Der Aufruf dieser Funktion dient der Rückgabe eventuell noch nötiger Daten für einen gültigen Abschluss des verschlüsselten Formates (z.B. Padding).
- Input/Output-Format:

Parameter	Type	Direction	Description
EncryptionSession_id	Session_ID	in	Der Session-Handle für den Verschlüsselungsvorgang.
Enc_Data	BYTE-Array_PTR	In/out	Zeiger auf einen Array in dem die verschlüsselten Daten gespeichert werden.
Enc_Data_Length	Int_PTR	In/out	In: Größe des Arrays auf den Enc_Data zeigt. Out: Anzahl der verschlüsselten Bytes, die in Enc_Data gespeichert sind.
	ReturnCode	return	„CSM_Encryption_OK“: Das Verschlüsseln war erfolgreich. „CSM_Invalid_Session“: Es wurde eine ungültige Verschlüsselsungs-Session angegeben. „CSM_Enc_Data_Too_Small“: Der Array in Enc_Data ist zu klein.

5. SECURE COMMUNICATION MODULE

Das Secure Communication Modul (SCM) dient als primäre Anlaufstelle für Applikationen oder Module der Middleware zur sicheren Kommunikation mit anderen Steuergeräten. Das SCM bietet Funktionen, die es erlauben sichere Kommunikationsbeziehungen zu entfernten Services auf- und abzubauen, sowie Daten über diese Beziehungen zu senden oder zu empfangen.

Um diese Services anbieten zu können, bedient sich das SCM der Services des Authentication Management-Modules (AMM) zur sicheren Authentifizierung der kommunizierenden Entitäten, des Key Management Modules (KMM) für eventuell nötigen Schlüsselzugang, sowie des Crypto-Services Modules (CSM) zur Durchführung der nötigen kryptographischen Berechnungen zur Kommunikationsabsicherung.

5.1 MOTIVATION

Die Schaffung von neuen Funktionen basiert in modernen eingebetteten Netzen oft auf der Vernetzung von Funktionalitäten auf unterschiedlichen Steuergeräten. Je nach Use Case haben die hierbei geschaffenen Kommunikationsbeziehungen unterschiedliche Schutzbedürfnisse. Diese reichen von komplett ungesichert, über die reine Übertragungssicherheit, der Authentizität und Integrität bis zur vollständigen Sicherstellung der Vertraulichkeit der Kommunikation. Es ist das Ziel von SEIS, Use-Case-bezogen die verlässliche Authentifizierung bzw. Identifikation der Kommunikationspartner und die Integrität der übermittelten Daten zu gewährleisten.

Die für die Kommunikations-Authenzität und –Integrität nötigen Verfahren beruhen zumeist auf kryptographischen Protokollen, deren korrekte Anwendung komplex und fehlerträchtig ist. Zu den Aufgaben des SCM gehört die Abstraktion dieser Implementierungsdetails gegenüber den Anwendungen. Applikationen sollen eine einfache und einheitliche Schnittstelle verwenden, die möglichst viel der techischen Komplexität verbirgt. Ziel ist, dass durch möglichst simple, applikationsunabhängige Methodenaufrufe eine abgesicherte Kommunikationbeziehung etabliert werden kann und möglichst viele Details auf Policing- oder Konfigurations-Ebene geklärt werden. Dies hat einerseits den Vorteil der einfacheren und somit schnelleren und weniger fehleranfälligen Anwendungsentwicklung, andererseits werden Voraussetzungen zur Kryptoagilität geschaffen, da Änderungen der kryptographischen Verfahren keine Anpassung der Anwendungen mehr bedingen.

5.2 VERFAHREN ZUR KOMMUNIKATIONSABSICHERUNG

Zur kryptographischen Absicherung der Kommunikation gibt es grundsätzlich zwei verschiedene Lösungswege. Einerseits gibt es generische Protokolle wie IPsec oder TLS, andererseits können auch speziell an die Kommunikation zwischen Steuergeräten angepasste Sicherungsverfahren Anwendung finden.

Zu den speziellen Techniken zählen u.a. die reine Absicherung der Header-Informationen von z.B. RPC-Paketen oder nur von speziellen Bereichen des Kommunikations-Payloads, die als besonders schützenswert erachtet werden. Diese spezialisierten Verfahren haben zwar den Vorteil, dass sie Daten sehr gezielt absichern und somit den kryptographischen Aufwand und Übertragungs-Overhead minimieren. Sie erfordern jedoch individuelle Lösungen (und sind somit fehleranfällig, kostenintensiv und nicht unbedingt interoperabel) und detaillierte Kenntnis von übergeordneten Protokollen und Anwendungen.

Bei den generischen Ansätzen IPsec oder TLS wird keine Differenzierung der konkreten Schützbedürfnisse der einzelnen Datenpakete vorgenommen sondern der gesamte Datenstrom einer Kommunikationsbeziehung kryptographisch abgesichert. Dies kann zwar einerseits zu höhererem (oder teilweise unnötigem) kryptographischen Aufwand führen, bietet jedoch den Vorteil der Absicherung ohne Wissen über die Semantik der Payload.

Desweiteren sind IPsec und TLS standardisierte und etablierte Absicherungsverfahren, deren Sicherheit und Zuverlässigkeit bereits eingehend geprüft wurde und zu denen robuste und effiziente Implementierungen existieren.

.

Die Implementierung der verwendbaren Absicherungsverfahren basiert auf einem Plugin-Modell, das die Sicherheitsmechanismen dem SCM transparent zugänglich macht. Hierdurch wird Krypto-Agilität unterstützt, da neue Mechanismen in Form von Plugins eingeführt werden können ohne weitere Teile des SCM oder gar der Middleware zu beeinträchtigen.

In SEIS wird das Verfahren IPsec gegenüber TLS und spezialisierten Lösungen favorisiert, da es ein bewährter und vielseitiger Standard ist, der die Kommunikation unterhalb des Transport-Layers des OSI-Modells absichert. Dies bietet den Vorteil, dass die Absicherung unabhängig von den darüber liegenden Protokollen und somit auch unabhängig von der gewählten Middleware-Lösung erfolgen kann.

Weiterführende Details zu den untersuchten IPsec-Details wie Schlüsselaustausch und angewandte Modi befinden sich in Abschnitt 4.2.2 „Schutz der internen Kommunikation“ in [14].

5.3 Wahl der verwendeten Kommunikationsabsicherung

Verfahren zur Kommunikationsabsicherung bieten zumeist eine Vielzahl von Optionen der Anwendung von technischen Mechanismen und kryptographischen Primitiven. Es ist die Aufgabe jedes Plugins diese Optionen einerseits durch Konfiguration oder andererseits durch Schnittstellen/API anzubieten.

IPsec bietet beispielsweise eine Vielzahl von Betriebsmöglichkeiten. Da es ein Standard ist, der die reine Absicherung der Kommunikationsdaten auf Basis von kryptographischen Schlüsseln umsetzt, erwartet IPsec die Bereitstellung dieser Schlüssel von anderer Stelle. Hiefür gibt es mehrere Möglichkeiten: über statisch vorkonfigurierte Schlüssel, sogenannten Pre-Shared-Keys (PSK) oder über dynamisch zwischen den kommunizierenden Entitäten ausgehandelten Session Keys. Für diesen dynamischen Schlüsselaustausch werden in SEIS die Verfahren der Familie des Internet Key Exchange Protocols (IKE) empfohlen.

Neben der oben bereits genannten Use-Case-spezifischen Schutzbedürfnisse gibt es somit weitere Optionen die konkret verwendeten Sicherungsverfahren festzulegen. Diese Wahlmöglichkeit bietet das SCM den kommunizierenden Entitäten in Form einer API an.

Die Festlegung der zu verwendenden Optionen eines Plugins kann über mehrere Wege geschehen. Zum einen über fixe Konfigurationen und Policies, zum anderen dynamisch nach Anforderung der aufrufenden Anwendung oder des Middleware-Moduls. Zur einfacheren Wartbarkeit des Middleware- und Applikations-Codes wird empfohlen diese Absicherungs-Charakteristika weitgehend über Policing festzulegen. Falls beides verwendet wird und ein Konflikt zwischen der Policy und der applikativen Anforderung auftritt, hat die Konfiguration der Policy Vorrang und der Verbindungsaufbau wird abgelehnt.

Die Wahl der anzuwendenden Policy Regeln erfolgt anhand der Ziel IP Adresse, des Ziel Ports und des gewünschten Kommunikationsverfahrens (verbindungsorientiert oder Datagramme). Eventuell nötige genauere Identifizierung der Kommunikationsbeziehung auf Service-Ebene erfolgt über das RPC-Modul der Middleware mit entsprechender dortiger Policy-Überprüfung.

5.4 Verbindungsfortsetzung

Ein Verbindungsaufbau über IPsec und IKE, inklusive dem Aushandeln von Schlüsseln kann zeit- und ressourcenaufwendig sein. Daher kann es Situationen geben wie beispielsweise in der Aufstartphase, in denen ein vollständiger Verbindungsaufbau nicht möglich ist und eine zeit-optimierte Lösung vorzuziehen ist.

Der IKE Standard bietet ein sogenanntes „Session Resumption“-Protokoll zur sicheren Wiederaufnahme von zuvor unterbrochenen Verbindungen. Hierbei wird der Zustand der Verbindung vor der Unterbrechung gespeichert, mit dem eigenen privaten Key verschlüsselt und an den jeweiligen Kommunikationspartner übertragen und lokal gespeichert. Bei der Wiederaufnahme der Kommunikationsbeziehung werden die Verbindungszustände wieder zurückgesendet, der Session-Key erneuert und die Verbindung fortgeführt. Der durch das Serialisieren des Verbindungs-Zustandes nötige Rechen- und Zeitaufwand ist bei den Nachlaufzeiten der betroffenen Steuergeräte zu berücksichtigen.

Bei welchen Verbindungen diese schnelle Wiederaufnahme möglich ist, entscheidet das Policy Management. Bei der Unterbrechung der Verbindung wird beim PDM angefragt, ob diese über Session

Resumption wiederherstellbar sein soll. Nur wenn diese Abfrage positiv beantwortet wird, darf der Verbindungszustand serialisiert und verschlüsselt an den Kommunikationpartner übertragen werden.

Will eine Applikation eine zuvor unterbrochene Verbindung wieder aufnehmen, kann es die Möglichkeit mit der Funktion „get_connection_state()“ erfragen. Diese Methode prüft, ob für die genannte Verbindung ein gespeicherter Zustand vorliegt. Da sich in der Zwischenzeit Änderungen bei den zulässigen Zertifikaten oder kryptographischen Verfahren ergeben haben können wird zudem beim PDM angefragt, ob diese Verbindung wiederaufgenommen werden darf.

Die Wiederaufnahme der Verbindung erfolgt über „resume_secure_connection()“.

5.5 AUFBAU EINER SICHEREN KOMMUNIKATIONSBEZIEHUNG

Um die interne Funktionsweise des SCM und dessen Zusammenspiel mit weiteren Sicherheitsmodulen zu verdeutlichen, wird nachfolgend der Prozess des Aufbaus einer sicheren Kommunikationsbeziehung beschrieben.

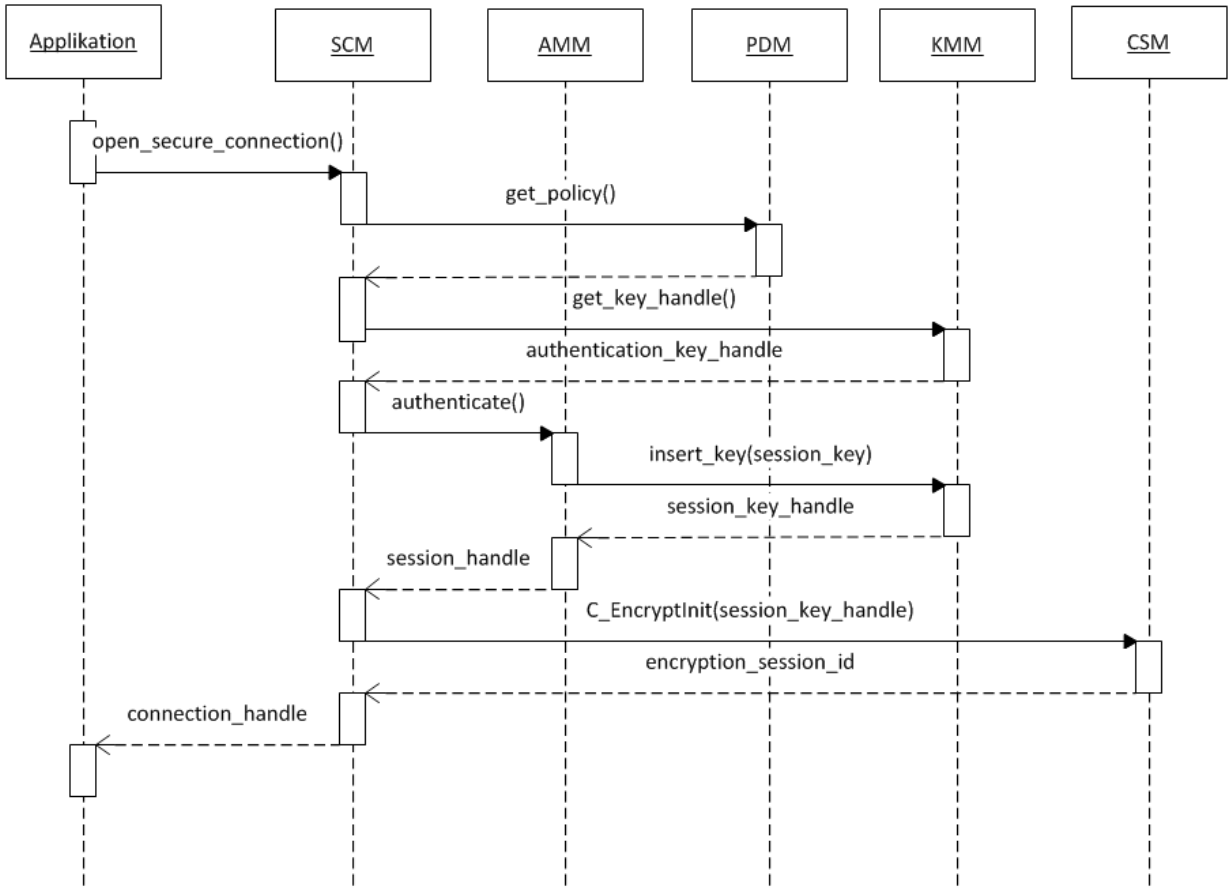

Abbildung 11: Aufbau einer sicheren Verbindung

Bei einem Aufruf von „open_secure_connection()“ werden zunächst die für die angeforderte Verbindung zutreffenden Policies vom PDM geholt. Diese Policies können z.B. Konfigurationen für die Verbindung beinhalten (welche Sicherungsverfahren werden angewandt, welche Schlüssel verwendet) oder die Verbindung explizit verbieten. Diese Policies werden mit den eventuell beim Aufruf von „open_secure_connection()“ übergebenen Parametern abgeglichen. Falls keine Widersprüche gefunden werden, stehen nun samtliche Konfigurationsdetails für die aufzubauende Verbindung fest.

Da ein Verbindungsaufbau eine erfolgreiche Authentifikation voraussetzt, wird ein Handle auf den zu verwendenden Authentifizierungsschlüssel beim KMM angefordert. Dieser wird dem AMM zur Durchführung der Authentifikation beim Aufruf von „authenticate()“ übergeben.

Während der Authentifikation kann ein Session-Key generiert werden oder der Authentifizierungsschlüssel als sog. Pre-Shared Key zur Nutzdatenabsicherung weiterverwendet werden. Der gewählte Schlüssel wird sodann per „insert_key()“ dem KMM mitgeteilt und ein Session-Key Handle erzeugt.

Mit diesem Session-Key Handle erzeugt das SCM mit dem Aufruf von „C_EncryptInit()“ eine Krypto-Session beim CM. Diese Krypto-Session dient dem Kodieren und Dekodieren der zu sendenden und empfangenen Daten. Hier werden auch die konkreten technischen Details des gewählten Sicherungsverfahrens spezifiziert.

Mit dem Erhalt des Verschlüsselungs-Session Handles gilt der Verbindungsaufbau als erfolgreich abgeschlossen und die aufrufende Anwendung erhält einen Verbindungs-Handle in Form eines SecureSockets zurück, über den sie die Verbindung bedienen kann.

5.6 ANWENDUNGSSPEZIFISCHES SECURITY API

Die vom SCM angebotene API lehnt sich an die POSIX API zur Socket-basierten an. Analog zu POSIX bietet die hier vorgestellte API Funktionen zum Aufbau von Kommunikationsbeziehungen, dem Senden und Empfangen von Daten und dem Abbau der Kommunikation. Identifiziert werden die offenen Kommunikationsbeziehungen durch SecureSocket_IDs.

Die API unterstützt die Verwendung von sowohl IPv4 als auch IPv6 Adressen. Dies geschieht über einen Transparenten Datentyp „IPAddress“. Dieser ist wie folgt definiert:

Feldname	Datentyp
IsIPv6Address	Boolean
RawIPAddress	Byte Array

Im Fall einer IPv4 Adresse ist die Länge von RawIPAddress 4 Bytes lang, im Fall von IPv6 beträgt die Länge des Arrays 16 Bytes.

Die oben beschriebenen Optionen für eine gesicherte Verbindung können über das Policing oder von der Anwendung direkt beim Aufruf von „openSecureConnection()“ festgelegt werden. Im Fall der Spezifikation über den Funktionsaufruf geschieht das in Form eines Arrays von Optionen. Dieser Array besteht aus Tupeln aus Optionsname und Optionswert. Nachfolgend eine expemplarische (unvollständige) Befüllung eines solchen Arrays:

Property-Name	Property-Value
KeyExchangeProtocol	IKEv2
AuthenticationCertificateID	1234567890
SessionResumptionAllowed	Yes

Im Folgenden wird auf die essentiellen Funktionen zum Erstellen, Betreiben und Schließen von sicheren Verbindungen sowohl als Client als auch als Server eingegangen. Auf eine Auflistung von asynchronen Funktionen wird verzichtet.

5.6.1 Funktion: open_secure_connection()

- Funktionsbeschreibung: Diese Methode erstellt eine sichere Verbindung zu einem gewünschten Kommunikationspartner, der einen Service anbietet. Mit Hilfe dieser Funktion wird ein Kommunikationskanal aufgebaut und ein Socket zurückgegeben, mit dem Daten über die Verbindung gesendet oder empfangen werden können. Der Funktionsaufruf enthält einen optionalen Array, mit dem detaillierte Kommunikations-Konfigurationen spezifiziert werden können. Wird der Array nicht angegeben, wird eine passende Standard-Konfiguration des Policing verwendet.
- Input/Output-Format:

Parameter	Type	Direction	Description
DestAddress	IPAddress	In	IP Adresse des Ziel-Steuergerätes oder Servers mit dem eine Verbindung aufgebaut werden soll.
Port	Integer	In	Port Nummer des Ziel Services mit dem eine Verbindung aufgebaut werden soll.
ConnectionType	Integer	In	Art der Verbindung: „Stream_Connection“: die Verbindung ist stream orientiert und stellt die Übertragung der Daten sicher. Dies entspricht „TCP“. „Datagram_Connection“: die Verbindung basiert auf nicht verbindungsorientieren Datagrammen. Dies entspricht „UDP“.
OptionArray	Option Array	In [Optional]	Array mit Optionen zur Spezifikation der angewandten Sicherheitsverfahren. Dieser Parameter ist optional. Bei Nicht-Angabe wird die Standard-Verbindungskonfiguration vom Policing verwendet.
ConnectionSocket	SecureSocketID	Out	Bei erfolgreichem Verbindungsaufbau wird hier der erstellte sichere Socket zurückgegeben.
	ReturnCode	return	„SCM_connection_established“: Der Verbindungsaufbau wurde erfolgreich abgeschlossen. „SCM_unauthorized_options“: Die angegebenen Verbindungs-Optionen wurden durch das Policing abgelehnt. „SCM_no_connection_configuration“: Das Policing hat keine Konfiguration für die angeforderte Verbindung und es wurden keine Optionen durch den Funktionsaufruf festgelegt. „SCM_invalid_options“: Die angegebenen Verbindungsoptionen sind nicht korrekt (schlüssig) oder unvollständig. „SCM_peer_unresponsive“: Der Kommunikationspartner antwortet nicht. „SCM_connection_rejected“: Der Kommunikationspartner hat einen Verbindungsaufbau abgelehnt. Dies kann sowohl an falscher Verbindungskonfiguration als auch an einer abgelehnten Authentifizierung liegen.

5.6.2 Funktion: resume_connection()

- Funktionsbeschreibung: Diese Methode erlaubt es eine bereits früher aufgebaute, jedoch unterbrochene sichere Verbindung wiederherzustellen. Zur Identifizierung der fortzusetzenden Verbindung wird ein Handle auf den alten SecureSocket benötigt.
- Input/Output-Format:

Parameter	Type	Direction	Description
ConnectionSocket	SecureSocketID	In	Der Socket Handle der wiederherzustellenden Verbindung.
	ReturnCode	return	„SCM_connection_resumed“: Die Verbindung wurde erfolgreich wiederhergestellt. „SCM_resume_no_session_state“: Es wurde kein passender Session State gefunden, der es erlaubt die Verbindung fortzusetzen. „SCM_resume_unauthorized“: Das Fortführen der Verbindung wurde vom Policing abgelehnt. „SCM_resume_rejected“: Das Fortführen der Verbindung wurde vom Kommunikationspartner abgelehnt. „SCM_peer_unresponsive“: Der Kommunikationspartner antwortet nicht.

5.6.3 Funktion: get_connection_state()

- Funktionsbeschreibung: Mit dieser Methode kann der Zustand der angegebenen Verbindung abgefragt werden.
- Input/Output-Format:

Parameter	Type	Direction	Description
ConnectionSocket	SecureSocketID	In	Der Handle der abzufragenden Verbindung.
ConnectionState	Integer	Out	Der Zustand der Verbindung: „SCM_connection_closed“: Die Verbindung ist geschlossen. „SCM_connection_established“: Die Verbindung ist etabliert und Daten können gesendet und empfangen werden. „SCM_connection_resumable“: Die Verbindung wurde unterbrochen, ist jedoch wiederherstellbar.
	ReturnCode	return	„SCM_connectionstate_retrieved“: Der Zustand der Verbindung konnte ermittelt werden. „SCM_invalid_socket“: Die angegebene Verbindung ist unbekannt.

5.6.4 Funktion: receive_data()

- Funktionsbeschreibung: Mit dieser Methode können Daten über eine bereits erstellte Kommunikationsbeziehung empfangen werden und in einem Empfangspuffer gespeichert werden. In

ReceivedBytesCount wird die Anzahl der erfolgreich empfangenen Bytes zurückgegeben. Ist diese Anzahl geringer als die Größe des Empfangspuffers, ist das ein Indiz für einen abgeschlossenen Empfangsvorgang. Die Funktion ist wiederholt aufzurufen bis der Empfangsvorgang abgeschlossen ist.

- Aufrufbeispiel:

```
unsigned char receiveBuffer[200];
int receivedBytes = 0;
receive_data(socket, receiveBuffer, sizeof(receiveBuffer),
&receivedBytes);
/* do sth with the data in receiveBuffer */
while (receivedBytes == sizeof(receiveBuffer)) {
    receive_data(socket, receiveBuffer, sizeof(receiveBuffer),
    &receivedBytes);
    /* do sth with the data in receiveBuffer */
}
```

- Input/Output-Format:

Parameter	Type	Direction	Description
ConnectionSocket	SecureSocketID	In	Der Socket, der die Verbindung identifiziert.
ReceiveBuffer	Byte Array	In	Puffer in den empfangene Bytes geschrieben werden sollen.
ReceiveBufferLength	Integer	In	Größe des Empfangspuffers.
ReceivedBytesCount	Integer	Out	Anzahl der empfangegenen Bytes.
	ReturnCode	return	„SCM_receive_successfull“: Die Daten wurden erfolgreich empfangen. „SCM_invalid_parameters“: Die angebenen Parameter sind ungültig. Dies kann z.B. eine nicht passende Länge des Empfangspuffers sein. „SCM_reception_failed“: Der Empfangsvorgang ist fehlgeschlagen. Dies kann auf einem Zusammenbrechen der darunterliegenden Kommunikationsbeziehung liegen.

5.6.5 Funktion: send_data()

- Funktionsbeschreibung: Mit dieser Methode können Daten über eine bereits erstellte Kommunikationsbeziehung gesendet werden. Es wird versucht die Daten als Ganzes zu versenden. In SentBytesCount wird die Anzahl der erfolgreich gesendeten Bytes zurückgegeben. Ist diese Anzahl geringer als die Größe des Sendepuffers, ist das ein Indiz für einen abgeschlossenen Sendevorgang. Die Funktion ist wiederholt aufzurufen bis der Empfangsvorgang abgeschlossen ist.
- Aufrufbeispiel:

```
unsigned char sendData[200];
int sentBytes = 0;
int totalSentBytes = 0;
send_data(socket, data[0], sizeof(sendData), &sentBytes);
totalSentBytes = sentBytes;
while (totalSentBytes < sizeof(data)) {
    send_data(socket, data[totalSentBytes], sizeof(sendData) –
    totalSentBytes, &sentBytes);
    totalSentBytes += sentBytes;
```

}

- Input/Output-Format:

Parameter	Type	Direction	Description
ConnectionSocket	SecureSocketID	In	Der Socket, der die Verbindung identifiziert.
DataToSend	Byte Array	In	Array der zu sendenden Daten.
DataToSendLength	Integer	In	Anzahl der zu sendenden Bytes.
SentBytesCount	Integer	Out	Anzahl der gesendeten Bytes.
	ReturnCode	return	„SCM_send_successfully“: Die Daten wurden erfolgreich gesendet. „SCM_invalid_parameters“: Die angebenen Parameter sind ungültig. „SCM_sending_failed“: Der Sendevorgang ist fehlgeschlagen. Dies kann auch an einem Zusammenbrechen der darunterliegenden Kommunikationsbeziehung liegen.

5.6.6 Funktion: close_connection()

- Funktionsbeschreibung: Diese Methode schließt bestehende Kommunikationsbeziehungen. Die hier angegebene SocketID ist anschließend ungültig und kann somit für keine weiteren Operationen verwendet werden.
- Input/Output-Format:

Parameter	Type	Direction	Description
ConnectionSocket	SecureSocketID	In	Der Socket, der die Verbindung identifiziert.
	ReturnCode	return	„SCM_socket_closed“: Der angegebene Socket wurde erfolgreich geschlossen. „SCM_invalid_socket_ID“: Der angegebene Socket ist ungültig oder bereits geschlossen.

5.6.7 Funktion: create_secure_socket()

- Funktionsbeschreibung: Diese Methode erstellt einen Server-Socket zum Anbieten eines Dienstes mit dem sich Clients verbinden können.
- Input/Output-Format:

Parameter	Type	Direction	Description
ConnectionType	Integer	In	Art der Verbindung: „Stream_Connection“: die Verbindung ist stream orientiert und stellt die Übertragung der Daten sicher. Dies entspricht „TCP“. „Datagram_Connection“: die Verbindung basiert auf nicht verbindungsorientieren Datagrammen. Dies entspricht „UDP“.
ServerSocket	SecureSocketID	Out	Bei erfolgreichem Verbindungsaufbau wird hier der erstellte sichere Socket zurückgegeben.
	ReturnCode	return	„SCM_connection_created“: Der Socket wurde erfolgreich erstellt.

5.6.8 Funktion: bind()

- Funktionsbeschreibung: Diese Methode bindet einen Server-Socket an eine Netzidentität und Sicherheitsmechanismen. Über wiederholte Aufrufe von „bind()“ kann der Server-Socket an merhere Netzidentitäten mit individuellen Sicherheitskonfigurationen gebunden werden. Der Funktionsaufruf enthält analog zu „open_secure_connection()“ einen optionalen Array, mit dem detaillierte Kommunikations-Konfigurationen spezifiziert werden können. Wird der Array nicht angegeben, wird eine passende Standard-Konfiguration des Policing verwendet.
- Input/Output-Format:

Parameter	Type	Direction	Description
ServerSocket	SecureSocketID	In	Der Socket, der die Verbindung identifiziert.
ServerAddress	IPAddress	In	IP Adresse der Netzidentität über welche der Service angeboten werden soll. Dies ist nötig für den Fall, dass das ausführende Steuergerät aufgrund von Multi-Homing mehrere Netzidentitäten (sprich IP Adressen) besitzt.
Port	Integer	In	Port Nummer der angebotenen Services.
OptionArray	Option Array	In [Optional]	Array mit Optionen zur Spezifikation der angewandten Sicherheitsverfahren. Dieser Parameter ist optional. Bei Nicht-Angabe wird die Verbindungskonfiguration vom Policing verwendet.
	ReturnCode	return	„SCM_socket_bound“: Der angegebene Socket wurde erfolgreich gebunden. „SCM_invalid_socket_ID“: Der angegebene Socket ist ungültig oder bereits geschlossen. „SCM_unauthorized_options“: Die angegebenen Verbindungs-Optionen wurden durch das Policing abgelehnt. „SCM_no_connection_configuration“: Das Policing hat keine Konfiguration für die angeforderte Verbindung und es wurden keine Optionen durch den Funktionsaufruf festgelegt. „SCM_invalid_options“: Die angegebenen Verbindungsoptionen sind nicht korrekt (schlüssig) oder unvollständig.

5.6.9 Funktion: listen()

- Funktionsbeschreibung: Mit dieser Methode wird der Server-Socket aktiviert und der Warteprozess für eingehende Verbindungen gestartet.
- Input/Output-Format:

Parameter	Type	Direction	Description
ServerSocket	SecureSocketID	In	Der Server-Socket der Verbindung
	ReturnCode	return	„SCM_listening_started“: Der Warteprozess wurde erfolgreich gestartet. „SCM_listening_failed“: Der Warteprozess konnte nicht getstartet werden.

5.6.10 Funktion: accept()

- Funktionsbeschreibung: Mit dieser Methode werden eingehende Client-Verbindungen akzeptiert. Dies ist eine blockierende Funktion, die erst zurückkehrt, wenn ein Client den Aufbau einer sicheren Verbindung mit erfolgreicher Authentifizierung durchlaufen hat.
- Input/Output-Format:

Parameter	Type	Direction	Description
ServerSocket	SecureSocketID	In	Der Server-Socket der Verbindung
ClientSocket	SecureSocketID	Out	Der Socket, der die Client-Verbindung identifiziert.
	ReturnCode	return	„SCM_client_accepted: Eine Client Verbindung wurde erfolgreich aufgebaut. „SCM_connection_rejected“: Der Verbindungsaufbau wurde (z.B. durch das Policing) abgelehnt. Dies kann sowohl an falscher Verbindungskonfiguration als auch an einer abgelehnten Authentifizierung liegen.

6. AUTHENTICATION MANAGEMENT MODUL

Die Authentifizierung ist eine zentrale Sicherheitsanforderung für jedes IT-System. Sie besteht allgemein darin, spezifische Eigenschaften (bspw. auch die Identität) einer entfernten Entität zu bestätigen. Die Authentifizierung kann entweder direkt (die Dienste, die kommunizieren wollen, führen den Authentifizierungsprozess selbstständig aus) oder indirekt sein (die Dienste vertrauen einer externen Entität, (Trusted Third Party – TTP), dort die Authentifizierungsprozesse durchzuführen) und wird von starken kryptographische Verfahren unterstützt.

6.1 ROLLEN DES AUTHENTIFIZIERUNGSMODULS

Das Authentication Management Modul ist für mehrere authentifizierungsrelevante Aufgaben notwendig. Die wichtigsten Rollen sind:

- Benutzer-, Dienste-, Geräte-Identifizierung und Authentifizierung;
- Authentifizierungsmanagement (Speicherung des Authentifizierungsstatus, Management der Authentifizierungstoken);
- Single-Sign-On/Single-Sign-Out Infrastruktur;
- Privacy-Mechanismen (bspw. Pseudonymierung der Identifizierung für Geräte, die an externer Kommunikation teilnehmen).

6.2 AUTHENTIFIZIERUNGSKONZEPT

Es gibt grundsätzlich 2 Arten der Authentifizierung: direkt und indirekt. Das System erlaubt jedoch auch die Single-Sign-On Möglichkeit.

6.2.1 Direkte Authentifizierung

Die direkte Authentifizierung findet zwischen 2 Diensten statt. Sie kann gegenseitig (beide Dienste führen eine Authentifizierung durch) oder einseitig (nur ein Dienst authentifiziert den anderen) sein. Sie besteht aus dezentralen Methoden und jeder Dienst muss das Protokoll implementieren (z.B. ein Authentifizierungshandshake). Die Dienste sollten in diesem Fall einen direkten Zugriff auf die Module haben, welche die kryptographischen Verfahren und Security-Policies verwalten.

Die direkte Authentifizierung wird bei den Protokollen IPsec+IKE, TLS/SSL und anderen Security-MW Protokollen mit Authentifizierungshandshake (z.B. ETCH [1], ICE-E, …) verwendet.

6.2.2 Indirekte Authentifizierung

Die indirekte Authentifizierung findet zwischen einem Dienst, sich authentisieren will, und einer Third-Party-Infrastruktur (TP), die für den Authentifizierungsprozess verantwortlich ist, statt. Nach dem Prozess erhält der Dienst ein Authentisierungsticket von der TP, der als Beleg für eine erfolgreiche Authentisierung dient. Außerdem berücksichtigt die TP das Autorisierungs-Management, um nur den gültigen und authentifizierten Diensten einen Zugriff zu einer Ressource zu geben. Die TP authentifiziert den Dienst oder nicht entsprechend einem Set von Security-Policies des Policy-Management-Modul, und vermerkt im Authentifizierungsticket, für welche Ressourcen es gültig ist.

Eine Master/Client Infrastruktur (z.B. EMVY [10]) oder eine Kerberos [15]-artige Infrastruktur verwenden solche Methoden.

6.2.3 Single-Sign-On Infrastruktur (SSO)

Die SSO Infrastruktur ist ein spezifischer Fall der indirekten Authentifizierung. Hierbei meldet ein Dienst sich einmalig zentral an der SSO Infrastruktur an und bekommt damit Zugriff zu jeder Ressource des Systems. Im Folgenden werden die diversen Vor- und Nachteile kurz dargestellt.

(+) System-Overhead;
(+) Abnahme der Vielfalt von unterschiedlichen Protokollen;
(+) Prozesszentralisierung (einfacheres Management);
(-) Komplizierte Rechteverwaltung und -beschränkung;
(-) Kompromittierung der SSO gibt dem Angreifer vollen Zugriff zum System.

Mit solchen Architekturen sind Single-Sign-Out Mechanismen notwendig, um die Ressourcenzugriffsrechte eines Dienstes systemweit zu löschen (z.B. Ticket Revocation List…).

6.3 MODULARCHITEKTUR

Für die direkte Authentifizierung ist das Modul dezentralisiert, jeder Dienst implementiert die Methoden zur Authentifizierung direkt..

Die indirekte und SSO Authentifizierung brauchen eine (trusted) Third Party, diese Architektur ist damit zentraler. Das Authentifizierungsmodul ist in einer einzelnen und zentralen Entität oder in einer verteilten hierarchischen Infrastruktur (redundante (und damit teurere) Architektur).

6.4 ANGRIFFSSZENARIEN

In dieser Sektion werden Angriffsszenarien auf unterschiedliche Ebene des AMMs dargestellt.

6.4.1 Angriff auf die Third-Party Infrastruktur

Eine zentrale TP-Infrastruktur für die Authentifizierung würde ein Single-Point-Of-Failure (SPOF) im System darstellen. Ein Systemausfall bspw. nach einem DoS Angriff würde eine kritische Auswirkung auf Sicherheit und Verfügbarkeit des Gesamtsystems haben, da kein Dienst mehr authentifiziert werden könnte. Ebenso muss die Kompromittierung der Infrastruktur als kritisch, angesehen werden, da hier entsprechend der Angreifer fälschlicherweise zentral authentifiziert werden könnte.

Daher müssen im System Gegenmaßnahmen bzw. Reaktionen vorgesehen werden:

- Detektion von Angriffen;
- Ersetzung der fehlerhaften/kompromittierten Infrastruktur durch
    - Eine Ersatzinfrastruktur (Verteiltes und Hierarchisches System);
    - Wechsel zu Mechanismen für eine direkte Authentifizierung;
    - Recovery-Modus mit reduzierter Sicherheit (ohne Authentifizierung).

Bei allen diesen Maßnahmen muss jedoch berücksichtigt und sichergestellt werden, dass Reaktionen und Fallback-Mechanismen nicht gezielt für einen Angriff ausgenutzt werden können, indem sie bspw. ein leichteres Angriffsziel bieten als die Originalarchitektur.

6.4.2 Angriff auf die Infrastruktur zur direkten Authentifizierung

Ein Angriff auf die Erzeugung der Policy/Authentication-Credentials (Zertifikate, Shared Secrets) würde als kritisch betrachtet (SPOF mit gleichen Gründen wie oben und Gegenmaßnahme mit einem verteilten System).

Ein direkter Angriff auf die Dienstauthentifizierung (DoS-, Impersonification-, Protocol-Tampering-Angriffe) würde abhängig von der Wichtigkeit des Dienstes eine beschränkte oder auch kritische Auswirkung haben.

6.5 FEHLERFÄLLE

In dieser Sektion werden Fehlerfälle für unterschiedliche Komponente des AMMs dargestellt.

6.5.1 Fehlerfall für die TP-Infrastruktur

Ein Ausfall dieser Infrastruktur würde als kritisch betrachtet (SPOF), da keine Dienstauthentifikation mehr möglich wäre. Ein Back-up System ist notwendig:

- Verteiltes und Hierarchisches System (Problem: Kosten, Redundanz, Management-Probleme);
- Zweiter Modus mit der Möglichkeit, die direkte Authentifizierung zu verwenden.

6.5.2 Fehlerfall für die direkte Authentifizierung

Ein Fehler im Authentifizierungsprozess würde als beschränkt oder kritisch betrachte. Jeder Ausfall betrifft die gegenseitige Authentifikation zweier Dienste, somit hängt die Auswirkung von ihrer Wichtigkeit ab. Für die Fehler Recovery existieren verschiedene Möglichkeiten:

- Die Dienste können den Authentifizierungsprozess mehrmals versuchen (Retry Lösung);
- Im Fall eines Fehlers in den Authentication-Credentials (Konfigurations-, Authentifizierungs-, Autorisierungs-Ticket, Zertifikat, Shared-Secret) können die Dienst eine (Re-)Synchronisation oder eine Erneuerung dieser Credentials nachfragen;
- Für den Fail-Safe-Modus können die Dienste die Authentifizierungsbedingung abschwächen (zB Authentifizierung mit alten Credentials anstatt den Letzten), allerdings müssen die Authentifizierungsmöglichkeiten zu ihrem Minimum begrenzt werden (dh nur die Notwendige Verbindungen zwischen Steuergeräte). Speziell hier ist allerdings sicherzustellen, dass das gezielte Auslösen dieses Modus nicht als Angriff verwendet werden kann.

6.6 ANWENDUNGSSPEZIFISCHES SECURITY API

Die Security API erlaubt dem Entwickler, die Authentifizierungskomponenten zu benützen:

- Um sich an einer TP-Infrastruktur anzumelden;
- Um neue Authentication-Credentials nachzufragen;
- Um Authentication-Credentials zu bestätigen;
- Um sich direkt einem anderen Dienst gegenüber zu authentisieren;
- Um einen anderen Dienst zu authentifizieren;
- Um die Konfiguration der Authentifizierung zu setzen oder zu verändern.

6.6.1 Funktion: AMM_TP_login()

- Funktionsbeschreibung: Diese Methode wird von einem Dienst verwendet, um sich auf der TP-Infrastruktur anzumelden. Der Dienst authentisiert sich auf der TP und kann später neue Authentication-Credentials oder die Bestätigung von Authentication-Credentials nachfragen.
- Input/Output-Format:

Parameter	Type	Direction	Description
TP_infra	Entity_ID	In	Identität der TP-Infrastruktur
Entity	Entity_ID	In	Identität des zu authentifizierenden Dienstes
SecData	Security_Metadata	In	Authentifizierungsmethode und Security-Credentials für Authentifizierung des Dienstes

Parameter	Type	Direction	Description
	ReturnCode	return	„Authentication_valid“: Der Authentifizierungsprozess war Erfolgreich, der Dienst ist jetzt angemeldet. „Authentication_invalid“: Der Prozess war erfolglos. „Authentiaction_pending“: Die TP benötigt weitere Schritte oder Token für den Authentifizierungsprozess. Der Dienst muss die Methode ein weiteres Mal mit anderen Authentication-Credentials aufrufen. „Method_Not_Supported“: Die Authentifizierungsmethode wird nicht unterstützt. Der Prozess war erfolglos. „Unknown_Security_Credential”: Die Authentication-Credential sind ungültig. Der Prozess war erfolglos.

6.6.2 Funktion: AMM_TP_logoff()

- Funktionsbeschreibung: Diese Funktion wird verwendet, um einen Dienst abzumelden. Die TP bestätigt das Vorhandensein des Dienstes in ihrer Datenbank, fragt nach der Autorisierung und löscht die Entität.
- Input/Output-Format:

Parameter	Type	Direction	Description
TP_infra	Entity_ID	In	Identität der TP-Infrastruktur
Entity	Entity_ID	In	Identität des abzumeldenden Dienstes
	ReturnCode	return	„Valid_Logoff“: der Prozess war erfolgreich. „Invalid_Logoff“: der Prozess war erfolglos.

6.6.3 Funktion: AMM_request_authentication_credential()

- Funktionsbeschreibung: Mit dieser Methode kann ein Dienst bei der TP neue Authentication-Credentials (Authentication-Ticket, Zertifikat…) anfordern. Die TP bestätigt die Anmeldung des Dienstes, fragt nach der Autorisierung und erstellt das Security-Credential.
- Input/Output-Format:

Parameter	Type	Direction	Description
TP_infra	Entity_ID	In	Identität der TP-Infrastruktur
Req_credential	Credential_type	In	Beschreibung des angeforderten Authentication-Ticket (für einem spezifischen Dienst, SSO…).
Authentication_Credential	Credential	Out	Das angeforderte Authentication-Credential oder ein Exception, wenn den Prozess erfolglos ist.
	ReturnCode	return	„Valid_request“: der Prozess war erfolgreich. „Invalid_request“: der Prozess war erfolglos.

6.6.4 Funktion: AMM_verify_authentication_credential()

- Funktionsbeschreibung: Mit dieser Methode kann ein Dienst oder die TP die Gültigkeit eines Authentication-Tickets bestätigen. Die TP bestätigt in ihrer Datenbank die Gültigkeit dieses Authentication-Credentials.
- Input/Output-Format:

Parameter	Type	Direction	Description
TP_infra	Entity_ID	In	Identität der TP-Infrastruktur, die das Credential bestätigen soll
Authentication_Credential	Security_Credenital	In	Zu bestätigendes Authentication_Credential.
	ReturnCode	return	„Valid_Authentication_Credential“: das Authentication-Credential ist gültig. „Invalid_Authentication_Credential“: das Authentication-Credential ist ungültig.

6.6.5 Funktion: AMM_start_authentication() & AMM_finish_authentication()

- Funktionsbeschreibung: Diese zwei Methoden werden für den Authentifizierungsprozess (direkt oder indirekt) zwischen zwei Diensten verwendet. Die zwei Dienste beginnen entweder den Authentication-Handshake oder, sie bestätigen die empfangenen die Authentication-Credentials.
- Input/Output-Format:

Parameter	Type	Direction	Description
Entity	Entity_ID	In	Identität des Dienstes, mit dem der initiale Dienst kommunizieren will.
Methode	Methode	In	Verwendete Authentifizierungsmethode (direkt, indirekt, mit TP…)
SecData	Security_Metadata	In	Authentication-, Authorization-Tickets, Security-Credentials, Shared-Secret…
	ReturnCode	return	„Authentication_valid“: Der Authentifizierungsprozess war erfolgreich, der Dienst ist jetzt angemeldet. „Authentication_invalid“: Der Prozess war erfolglos. „Authentiaction_pending“: Die TP benötigt weitere Schritte für den Authentifizierungsprozess. Der Dienst muss noch die Methode erneut mit anderen Authentication-Credentials aufrufen. „Method_Not_Supported“: Die Authentifizierungsmethode wird nicht unterstützt. Der Prozess war erfolglos. „Unknown_Security_Credential”: Die Authentication-Credential sind ungültig. Der Prozess war erfolglos.

6.6.6 Funktion: set_authentication_configuration()

- Funktionsbeschreibung: Diese Methode wird vom Autorisierungsmodul verwendet. Damit kann dieses Modul direkt auf der TP oder auf dem Dienst die Konfiguration der Authentifizierung einstellen. Das Autorisierungsmodul braucht einen sicheren Kanal zur TP oder zum Dienst.
- Input/Output-Format:

Parameter	Type	Direction	Description
Target	Entity_ID	In	Identität des Zieles (TP oder Dienst)
Config_Set	Configation_Ticket	In	Beschreibung der Methoden und Parameter für den Authentifizierungsprozess
	ReturnCode	return	„Setting_changed“: der Prozess war erfolgreich. „Setting_not_changed“: der Prozess war erfolglos.

6.7 SECURITY PLUG-IN

6.7.1 Credential-Maker Plug-in

Diese Schnittstelle wird als Plug-in implementiert, sie beschäftigt sich mit der Erstellung aller Authentication-Credentials (Authentication-, Attestation-Tickets, Zertifikate, Pseudonyme…). Das Plug-In wird von der Funktion request_authentication_credential() aufgerufen. Er realisiert eine sichere Ticketerstellung:

- Es baut einen sicheren Kanal zum Policy-Management-Modul (über das Sichere-Kanäle-Modul) auf und erfragt bei der Policy-Datenbank, ob der aufrufende Dienst die Berechtigung für die angefragten Tickets hat.
- Es baut einen sicheren Kanal zum Krypto-Modul (über das Sichere-Kanäle-Modul) auf und fordert die Erstellung eines neuen Tickets an.
- Es kümmert sich auch um die sichere Speicherung dieses Tickets in der geeigneten Ticket-Datenbank (z.B. Möglichkeit, den Schlüssel in dem Secure Hardware zu speichern).

6.7.2 Credential-Verifier Plug-in

Diese Schnittstelle wird als Plug-in implementiert, sie beschäftigt sie mit der Bestätigung aller Authentication-Credentials (Authentication-, Attestation-Tickets, Zertifikate, Pseudonyme…). Das Plug-In wird von der Funktion verify_authentication_credential() aufgerufen. Es realisiert eine sichere Ticketbestätigung:

- Es baut einen sichereen Kanal zum Policy-Management-Modul (über das Sichere-Kanäle-Modul) auf und bei der Policy-Datenbank, ob der aufrufende Dienst die Berechtigung für die angefragten Tickets hat
- Es baut einen sicheren Kanal zum Krypto-Modul (über das Sichere-Kanäle-Modul) auf und fragt nach der Bestätigung der Signatur (oder des Hash-Codes bzw. HMAC) des Tickets.
- Es übermittelt die Antwort zur anfragenden Entität (SKM, Anwendung…).

6.7.3 Internal Service Authentication Manager Plug-in

Dieses Plug-in ist verantwortlich für die Authentifizierungsbuchhaltung. Es überwacht jeden Schritt der Authentifizierungen, die vom Policy-Manager angefordert werden. Dieses Plug-in wird aufgerufen, wenn das Sichere-Kanal-Modul den Authentifizierungsprozess eines Dienstes beginnt. Es erzeugt einen Eintrag in der Datenbank und überwacht den Austausch mit der Krypto-Dienstleistungen. Außerdem aktualisiert es in einer Authentifizierungs-spezifischen Datenbank die Authentifizierungsschritte zwischen allen Diensten und die entsprechenden Authentifizierungstoken.

6.7.4 External Service Authentication Manager Plug-in

Die interne Kommunikation unterliegt oft starken Zeit- und Sicherheitsbeschränkungen, die die ECUs daran hindern, dynamische und direkte Verbindungen von externen Geräten zu akzeptieren. Der Sicherheits-Proxy spielt eine wesentliche Rolle, um externe Dienste zu integrieren, und um die Informationen an die interne Ziel-ECU (bei eingehender Kommunikation) bzw. an die externen Zielgeräte (bei ausgehenden Nachrichten) nach der Durchsetzung von notwendigen Sicherheitsmechanismen weiterzuleiten. Der

Proxy führt die Entkopplung der Sicherheitsmechanismen zwischen externen und internen Systemen aus. Das interne System ist also fähig, basierend auf seinen eigenen Kommunikations-Policies zu kommunizieren, obwohl die Daten an die Außenwelt unter Verwendung anderer Sicherheitsmechanismen und Protokolle übermittelt werden.

Das Authentifizierungs-Modul implementiert auf dem Proxy ein Plug-in für die Authentifizierungsverwaltung von externen Geräten:

- Authentication Accounting: das Plug-in ist fähig, die verschiedenen Stufen der Authentifizierung zu überwachen und den Status des Authentifizierungsprozesses für jede externe Entität zu speichern.
- Ausführung der Protokolle: Das Plug-in ist verantwortlich für die Sicherheit des Verbindungsaufbaus und koordiniert die Sicherheits-Plug-ins für den Kanalaufbau (zB SSL / TLS-Plug-in des SCM).
- Key-Management und Crypto-Services: Das Plug-in kann auf die verfügbare Sicherheitsinfrastruktur angewiesen sein. Das Plug-in bspw. einen neuen Session-Key für eine externe Kommunikation am KMM anfordern. Dieser Schlüssel wird wie jeder andere Kommunikationsschlüssel sicher vom HSM für Crypto-Services verwendet und gespeichert.
- Policy Management: das Plug-in muss andere Policies verwenden als diejenigen, die für interne Kommunikation verwendet werden. Diese Policies werden dynamisch vom Plug-in erstellt und berücksichtigen die angeforderten Authentifizierungsprozesse und eine Risikoanalyse des Kontextes. Die Policies werden später wie normale Policies gespeichert (e.g. im Proxy-HSM) oder auf sie zugegriffen.

Nach einer erfolgreichen Authentifizierung ermittelt der Proxy für eingehende Informationen zusätzliche Sicherheitsparameter, damit betroffene Steuergeräte eine granulärere Zugriffskontrolle durchsetzen können. Für ausgehende Informationen kann der Proxy zusätzliche Sicherheitsparameter aus den Steuergeräten bekommen, auch um die Autorisierung granularer zu machen.

6.7.5 Pseudonym Manager

Schutz der Privatsphäre ist eine wichtige Security-und Privacy-Anforderung in für C2X-Kommunikation. Künftige kooperative dezentrale Safety-Anwendungen verwenden mehr und mehr Fahrzeug-Heartbeat-Nachrichten, um eine aktuelle Position für jedes teilnehmende Auto zur Verfügung zu stellen. Die ersten Anwendungen wurden ohne explizite Berücksichtigung von Privacy-Aspekten entwickelt und erlauben passiven Angreifern, diese Daten zu belauschen [8]. Diese Nachrichten können Information enthalten über die räumliche Position des Fahrzeugs, aber auch – insbesondere indirekt über Korrelation und Fusion mit anderen Datenquellen - über den Fahrer selbst, mit Bezug zu seinem persönlichen Leben, sein Fahrzeug und eventuell mögliche Sicherheitsschwachstellen. Der Ansatz dieses Plug-in ist das Management einer Pseudonym-Infrastruktur:

- Erstellung und Verwaltung der Pseudonyme;
- Kommunikation und Zusammenarbeit mit anderen Fahrzeugen und Infrastruktur, um unterschiedliche Gruppen von Pseudonymen zu veröffentlichen oder ggf. zu widerrufen (Revocation);
- Management der Pseudonym-Security (Wechsel, anwendung und Revocation).

7. KEY-MANAGEMENT

Für das Key Management ist das Key Management Module (KMM) des Security Frameworks zuständig. Seine Aufgabe ist weiteren Elementen der Middleware, aber auch Applikationen Zugang zu kryptographischen Schlüsseln zu ermöglichen. Im Folgenden wird kurz auf die Motivation und die zu verwaltenden Schlüsselarten eingegangen. Danach werden zentrale und dezentrale Lösungen des Key Managements beschrieben und Möglichkeiten zur Absicherung der Abläufe innerhalb des KMMs erläutert.

7.1 MOTIVATION

Um die Kommunikation von Services, die auf unterschiedlichen ECUs implementiert sind, abzusichern, sollen verschiedene Lösungsmöglichkeiten durch die Middleware bereit gestellt werden, aus denen entsprechend den Bedürfnissen der Services ausgewählt werden kann. Die Kontrolle über die zu verwendende Sicherungslösung obliegt der Middleware der jeweiligen ECU. Sie ist zuständig für den Auf- und Abbau von sicheren Kommunikationskanälen sowie für die verwendeten Sicherungsmaßnahmen.

Viele der hierfür möglichen Lösungen (z.B. IPSec, TLS, etc.) basieren auf kryptographischen Verfahren, welche die Verwendung von Schlüsselmaterial erfordern. Dieser Abschnitt befasst sich mit möglichen Ansätzen zur Bereitstellung von Schlüsselspeichern für die Middleware der jeweiligen ECUs.

7.2 ARTEN VON SCHLÜSSELN

Grundsätzlich sind zwei Arten von Schlüsseln zu unterscheiden: temporäre und permanente Schlüssel.

Temporäre Schlüssel besitzen eine relativ kurze Gültigkeit und werden in regelmäßigen Abständen neu erzeugt (z.B. Session-Keys). Diese können von jeder ECU für die eigenen Kommunikationsbeziehungen selbst oder von einer zentralen ECU für alle anderen ECUs erzeugt werden. In beiden Fällen ergibt sich hierdurch der Bedarf für ein Verteilungssystem für diese Schlüssel. Hierfür können bereits etablierte Verfahren eingesetzt werden wie z.B. IKE. Die Schutzbedürfnisse für diese Art von Schlüsseln sind relativ gering, da sie keine lange Gültigkeitsdauer besitzen.

Permanente Schlüssel besitzen eine relative lange Gültigkeitsdauer und bieten sich vornehmlich für Schlüssel an, die nicht regelmäßig über ein Verteilungssystem neu zugeteilt werden können. Dies trifft vor allem z.B. bei Schlüsseln zur Authentifizierung von Kommunikation zwischen Fahrzeug- und OEM-Systemen zu. Meist werden langlebige Schlüssel zur Authentifizierung der Kommunikationpartner bei Schlüsselaustausch-Protokollen für kurzlebige Schlüssel verwendet. Da permanente Schlüssel hiermit eine wichtige Basis für die Sicherheit des Gesamtsystems bilden und das Ersetzen aufwendig ist (Konfigurationseingriffe, Firmware-Updates, Revocation-Lists), ist deren Schutzbedarf wesentlich höher als bei temporären Schlüsseln.

7.3 EIGENSCHAFTEN VON SCHLÜSSELN

Schlüssel besitzen neben ihren reinen kryptographischen Schlüsseldaten weitere Eigenschaften. Hierunter fallen einerseits beschreibende Daten sowie die Spezifikation der Zugriffs- und Nutzungsrechte. Die nachfolgende Aufzählung verdeutlicht exemplarisch, welche Arten von Eigenschaften ein Schlüssel haben kann. Sie erhebt keinen Anspruch auf Vollständigkeit.

Beschreibende Eigenschaften:

- Zu Grunde liegendes kryptographisches Verfahren (inkl. Sicherheitslevel bzw. Schlüssellänge)
- Gültigkeitszeitraum

Zugriffsrechte:

- Der Schlüssel darf (nicht) aus dem Security Framework herausgegeben, bzw. nicht oder nur unter definierten Bedingungen migriert werden (➔ get_key() scheitert)
- Bei zentral auf der KD_ECU gespeicherten Schlüsseln:
    - Zugriffsrechte anfragender ECUs, welche ECU erhält Zugriff auf den Schlüssel bzw. welche Authentifizierungs- und Autorisierungs-Token sind für den Zugriff nötig.
    - Der Schlüssel darf die KD_ECU (nicht) verlassen

Verwendungsrechte:

- Mit dem Schlüssel darf signiert werden (oder ein kryptographischer MAC erstellt werden).
- Mit dem Schlüssel dürfen Signaturen (oder MACs) überprüft werden.
- Mit dem Schlüssel darf verschlüsselt werden.
- Mit dem Schlüssel darf entschlüsselt werden.

Sollte der Schlüssel auf einem Zertifikat basieren, bzw. seine Gültigkeit durch ein Zertifikat belegt werden, können die Daten des Zertifikats verwendet werden um Teile der Eigenschaften zu spezifizieren.

7.4 SCHLÜSSELZUGRIFF

Wie in Kapitel 1 beschrieben, verwendet die Middleware den sogenannten „Security Abstraction Layer“ zur Erfüllung der Schutzbedürfnisse der beanspruchten Kommunikationsbeziehungen. Dieser Layer kontrolliert den Verbindungsaufbau und damit das Einrichten der sicheren Kommunikationskanäle. Hierfür ist ein Zugriff auf die benötigten Schlüssel notwendig. Dieser Zugriff erfolgt generell indirekt über eine Schnittstelle namens „get_key_handle()“ des Subsystems der lokalen Schlüsselverwaltung. Diese Funktion lädt den Schlüssel und gibt einen Zugriffs-Handle zurück. Dieser Handle kann im Krypto-Modul verwendet werden um Operationen mit dem Schlüssel auszuführenIm Folgenden werden Varianten beschrieben, wie ein Schlüsselverwaltungssystem umgesetzt werden kann.

7.4.1 Lokale Schlüsselspeicherung

Die erste Variante ist, dass Schlüssel lokal auf den ECUs gespeichert werden. Damit würde ein Aufruf von „get_key_handle()“ oder „get_key()“ lediglich den verlangten Schlüssel von einer lokalen Quelle laden und dem Security-Stub der Middleware liefern. Es gibt mehrere Stufen zur Absicherung der lokalen Speicherung:

- In Flash-Bausteinen auf der ECU. Auf den ECUs sind meist bereits Flash-Bausteine verbaut, welche die Firmware und Konfigurationsdaten der jeweiligen ECU bereitstellen. Die Speicherung im Flash bietet schwächere Sicherungsmöglichkeiten vor unbefugtem Zugriff auf sicherer Hardware-basierende Speicher, da entweder die Daten unverschlüsselt abgelegt werden, oder, falls verschlüsselt, die zum Entschlüsseln nötigen Werte entweder lokal (also unsicher) hinterlegt oder von einer anderen ECU geholt werden müssen. Die Methode der Speicherung im Flash ist als kostengünstig anzusehen, da hierfür keinerlei zusätzliche Voraussetzungen an die Hardware-Ausstattung der ECUs gestellt werden.
- Im Flash-Speicher, der in die CPU der ECU integriert ist. Es gibt die Möglichkeit CPUs direkt mit integrierten Speichern auszustatten. In der Regel ist die Kapazität dieser Lösung deutlich begrenzter als bei externen Flash-Speichern (bis zu 1 MB). Die Sicherheit der Daten in diesen Speichern ist prinzipiell ähnlich zu externen Speichern, jedoch ist der Zugang aufgrund der Integration in die CPU deutlich erschwert. Ein Angreifer müsste die CPU mechanisch öffnen, um Zugriff auf die Speicherinhalte zu bekommen.
  Man kann diesen internen Speicher jedoch mit externem Flash Speicher kombinieren: die zu schützenden Daten werden extern verschlüsselt abgelegt und der für den Zugriff nötige Schlüssel im internen Speicher sicher verwahrt. Hieraus ergibt sich ein ähnlicher Ansatz wie bei der Ver-

wendung von TPMs, bei denen aufgrund des sehr beschränkten internen Speichers die zu schützenden Daten extern verschlüsselt vorgehalten werden.

- Verbau von sicheren Hardware-Elementen wie z.B. einem TPM oder HSM. Bei einer Hinterlegung in einem sicheren Speicher bietet sich die Möglichkeit an, den Zugriff auf den sicheren Speicher nur zuzulassen, wenn die ECU dem Speicher nachweisen kann, dass sie unkompromittiert ist. Ein mögliches Verfahren hierzu ist die Durchführung eines Plattform-Integritäts-Tests beim Aufstart der ECU. Der Einsatz eines sicheren Speichermoduls stellt jedoch einen zu berücksichtigenden Kostenfaktor dar, da entweder ein separates Modul an die ECU angebunden oder in die Schaltungslogik des Hauptprozessors (CPU) des Steuergeräts integriert werden muss.

7.4.2 Zentrale Schlüsselspeicherung

Neben der lokalen Schlüsselspeicherung gibt es noch die Möglichkeit ein zentrales System zur Schlüsselverwaltung einzusetzen. Hierbei fungiert eine spezielle ECU als Key-Distribution System für eine Menge von Client-ECUs. Diese Menge kann eine Teilmenge aller ECUs sein, d.h. ECUs können sowohl Schlüssel zentral beziehen als auch lokal verwahren.

Die Key-Distribution-ECU (KD-ECU) besitzt einen sicheren Speicher zur Aufbewahrung der Schlüssel. Als Schutzmaßnahme des sicheren Speichers wird empfohlen den Zugriff erst bei erfolgreicher Validierung der KD-ECU freizugeben, etwa durch einen Plattform-Integritäts-Test.

Falls ein Modul der Middleware (oder eine Applikation, falls sie die Rechte hierfür hat) durch den Aufruf von „get_key()“ oder „get_key_handle()“ einen Schlüssel anfordert, wird ein Request an die KD-ECU gesendet, die dann die Gültigkeit der Anfrage validiert und den geforderten Schlüssel liefert. Diese Validierung fordert eine gegenseitige Authentifizierung von Client-ECU und KD-ECU und prüft ob die Client-ECU autorisiert ist, auf den angeforderten Schlüssel zuzugreifen. Jede Client-ECU hat nur auf die ihr zugewiesene Schlüssel Zugriff. Bei unbefugten Zugriffen kann die KD-ECU entsprechende Reaktionen einleiten wie z.B.

- Protokollieren des Fehlzugriffs
- Informieren eines evtl. vorhandenen Intrusion Detection Systems (IMM)
- Sperren jeglichen Zugriffs der anfragenden ECU auf die KD-ECU
- Benachrichtigung aller ECUs über das Fehlverhalten der möglicherweise kompromittierten ECU
- Informieren des Fahrers oder OEMs.

Bei erfolgreicher Validierung wird der angeforderte Schlüssel aus dem sicheren Speicher geholt und an die Client-ECU übertragen.

Ein wichtiger Punkt der hier zu berücksichtigen ist, ist der Schutz der Kommunikation zwischen Client-ECU und KD-ECU. Da hier kryptographische Schlüssel übertragen werden, muss diese Kommunikation über einen sicheren Kanal erfolgen, der authentisch, integer und vertraulich ist. Die Absicherung des Kanals kann über ausgehandelte Session-Keys realisiert werden. Für die hierfür nötige Authentifizierung sind einerseits potentiell unsicher gespeicherte lokale Schlüssel oder andererseits hardware-gestützte Techniken wie Physically Uncloneable Functions (PUFs) denkbar.

Ein wichtiger Vorteil des zentralen Schlüsselspeichers ist, dass es nur einen einzenen sicheren Speicher für viele ECUs des Systems gibt und sich somit der Einsatz von lokalen sicheren Speichern minimieren lässt. Dies kann zu erheblichen Kostenersparnissen bei der Umsetzung einer Sicherheitsarchitektur in Fahrzeugen führen.

Zudem ergibt sich eine Vereinfachung der Schlüsselverteilung. Einerseits gibt es nur noch eine zentrale ECU, auf der wichtige permanente Schlüssel (OEM-Keys) gespeichert werden. Andererseits bietet sich die KD-ECU an, um sämtliche temporären Schlüssel zu erzeugen (bspw. unter Verwendung eines besonders leistungsfähigen TRNG) und mittels der GetKey()-Anfragen der Client-ECUs an die entsprechenden ECUs zu verteilen.

Ein weiterer Vorteil liegt in der vereinfachten Wartbarkeit durch die zentrale Ablage der OEM-Schlüssel. Bei OEM-seitigem Wechsel der Schlüssel muss nur noch ein Firmware-Update der KD-ECU durchgeführt werden anstelle eines Updates jeder einzelnen betroffenen ECU.

Nachteile liegen in der zusätzlichen Komplexität des Schlüsselsystems durch die mehrstufige Absicherung der Schlüsselanfragen und der grundsätzlichen Kompromittierbarkeit von auf den Client-ECUs lokal gespeicherten Schlüsseln.

7.4.3 Beschreibung des Schlüsselzugangs

Die oben beschriebenen Konzepte der lokalen und zentralen Schlüsselspeicherung können im KMM kombiniert werden. Dies bedeutet, dass man je nach Kritikalität des jeweiligen kryptographischen Schlüssels die Wahlmöglichkeit hat ihn lokal auf der ECU zu speichern oder zentral in einer speziell gesicherten KD-ECU zu hinterlegen.

In den folgenden Abbildungen werden die Abläufe dieser kombinierten Lösung exemplarisch anhand der „get_key()“ Methode veranschaulicht. In Abbildung 12 werden die Abläufe dieser kombinierten Lösung exemplarisch anhand der „get_key()“ Methode veranschaulicht. Abbildung 12 zeigt die entsprechenden Abläufe auf der KD-ECU.

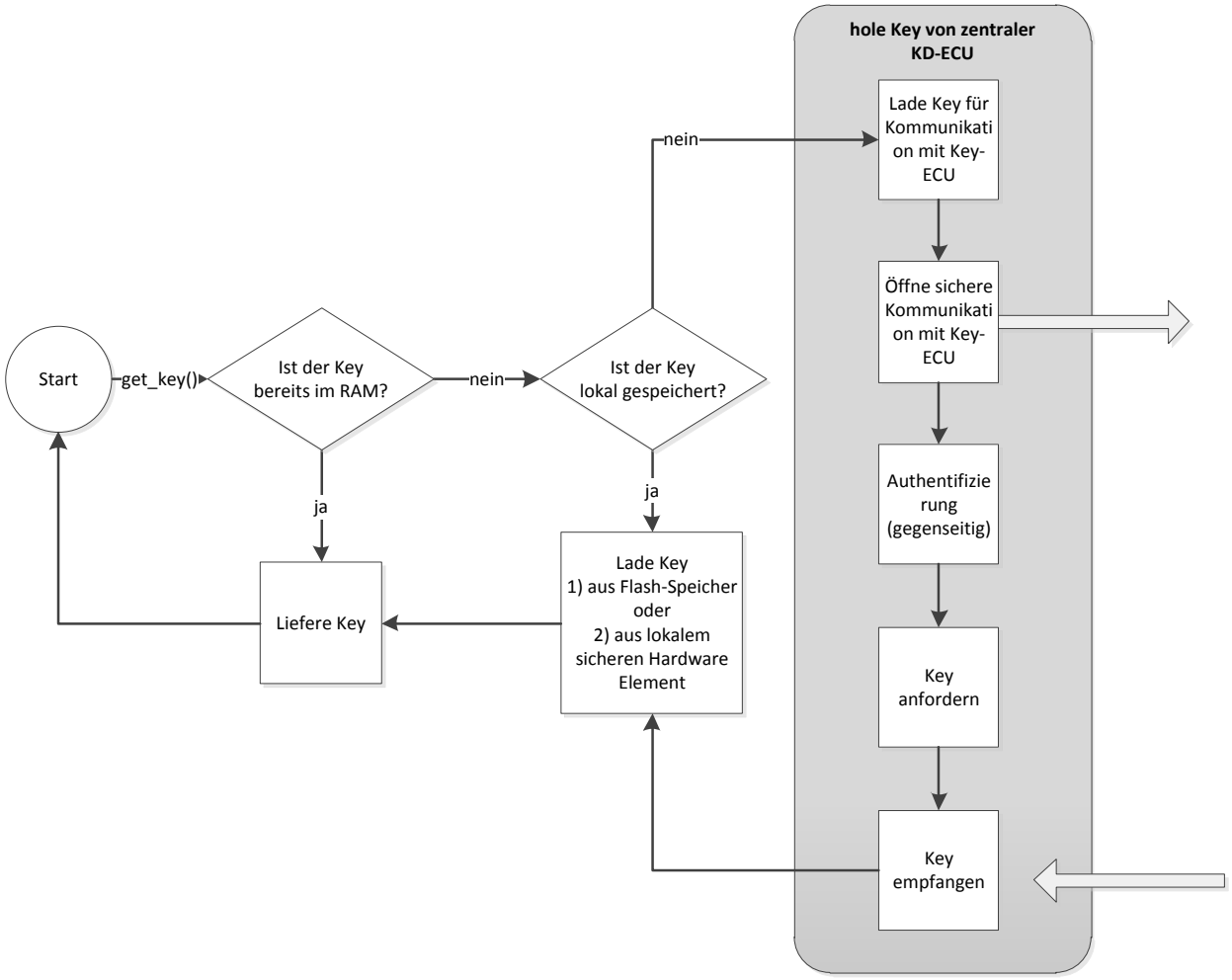

Abbildung 12: Ablauf der GetKey()-Methode der Client-ECU bei der Verwendung eines zentralen Schlüsselspeichers

Bei einem Aufruf der „get_key()“ Methode wird zunächst geprüft, ob der angefragte Schlüssel bereits im lokalen RAM verfügbar ist. Falls dies nicht der Fall ist, muss der Schlüssel geladen werden. Wenn der Schlüssel lokal gespeichert ist, holt ihn das KMM aus dem Flash-Speicher oder (falls vorhanden) aus einem sicheren Hardware Element Ist das Schlüssel nicht lokal vorhanden, muss er bei der KD-ECU angefragt werden. Hierzu ist zunächst ein sicherer Kanal zur KD-ECU aufzubauen. Die hierfür nötigen Authentifizierungsschlüssel müssen lokal verfügbar sein und geladen werden. Nach erfolgreicher gegenseitiger Authentifizierung und Etablierung des sicheren Kommunikationskanals wird der gewünschte Schlüssel bei der KD-ECU angefordert.

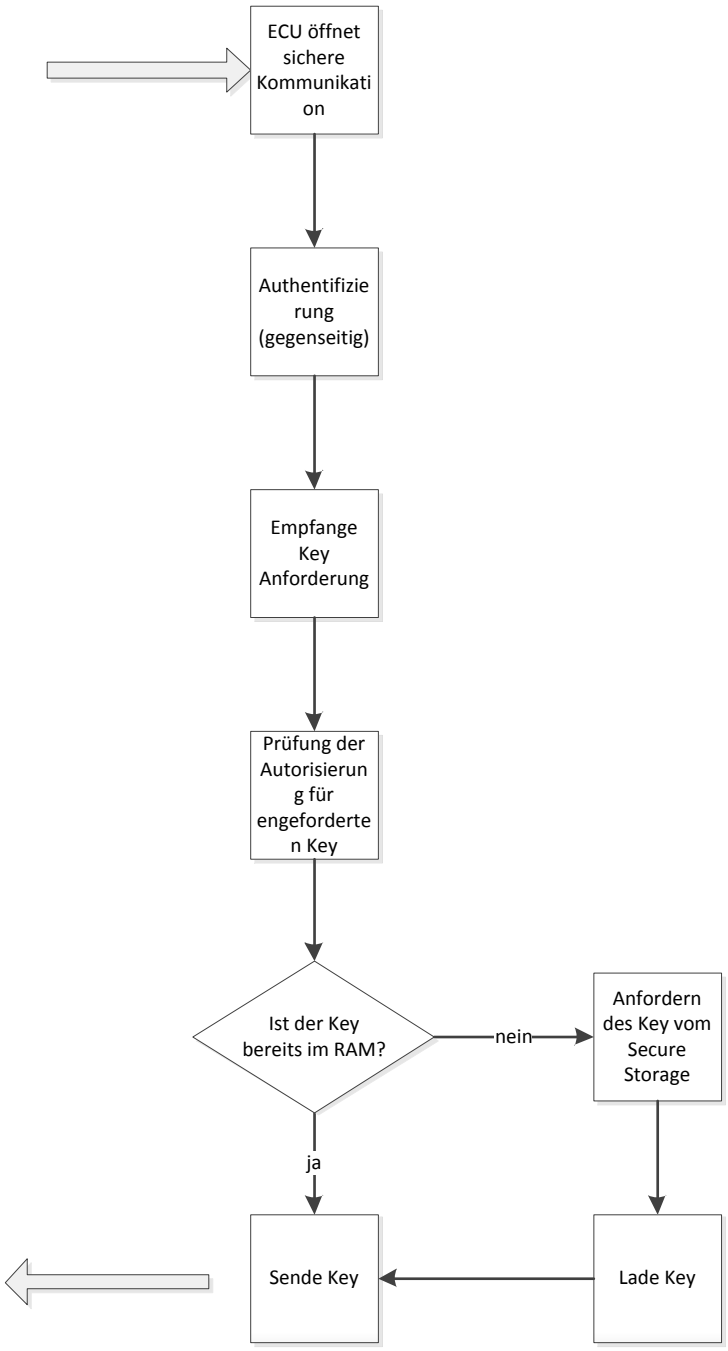

Abbildung 13: Ablauf der GetKey()-Methode der KD-ECU bei der Verwendung eines zentralen Schlüsselspeichers

Auf Server-Seite behandelt die KD-ECU die Requests der anfragenden ECUs. Hierzu wird analog zur Client-Seite ein sicherer Kommunikationskanal mit aufgebaut und die Schlüssel-Anfrage empfangen. Anschließend wird die Autorisierung der Anfrage geprüft, d.h.ob die anfragende ECU Zugriff auf den angeforderten Schlüssel erhalten darf. Anschließend wird der Schlüssel direkt ausgehändigt oder zunächst aus dem Secure Storage des Steuergeräts (sicheres Hardware Element) geladen.

7.5 ANWENDUNGSSPEZIFISCHES SECURITY API

Die API dienst als Schnittstelle für Module des Security Frameworks, weitere Module der Middleware sowie für Applikationen. Über die API werden folgende Dienste des Key Managements angeboten:

- Zugang zu kryptographischen Schlüsseln;
- Schlüsselverwaltung:
    - Aufnahme neuer Schlüssel
    - Löschen von Schlüsseln
    - Setzen/Update von Schlüsseleigenschaften
    - Schlüssel wiederrufen (Revocation)

Die oben beschriebenen Eigenschaften von Schlüsseln werden über Property Tabellen dargestellt. Diese Tabellen bestehen aus Zuweisungen von Property-Name und Property-Wert. Nachfolgend eine exemplarische (unvollständige) Befüllung einer solchen Tabelle:

Property-Name	Property-Value
KryptographicAlgorithm	AES 256
NotValidAfter	2016-12-31
KeyMayLeaveSecurityFramework	No
KeyCanSign	Yes

7.5.1 Funktion: get_key()

- Funktionsbeschreibung: Diese Methode erlaubt Zugriff auf kryptographische Schlüssel. Sie kann von jedem Modul oder jeder Applikation aufgerufen werden, welche Schlüssel benötigen. Diese Funktion führt transparent die Schritte aus obigen Abbildungen aus und gibt den Schlüssel zurück. D.h es wird versucht den Schlüssel von einer lokalen Quelle oder von einer zentralen KD-ECU zu laden. Entsprechend nötige Überprüfungen der Berechtigung werden beim Policy Management durchgeführt.
  Bemerkung: Diese Funktion dient dem direkten Ausliefern von kryptographischen Schlüsseln! D.h. sobald die Schlüssel an die anfragende Instanz übergeben worden sind obliegt es ihrer Verantwortung deren Sicherheit zu gewährleisten. Es wird empfohlen die Methode „get_key_handle()“ zu verwenden und mit Hilfe dieser Handles über das Kryptomodul Operationen auszuführen. Diese Methode sollte nur in Ausnahmefällen verwendet werden und die Schlüssel benötigen hierfür die nötige Freigabe.
- Input/Output-Format:

Parameter	Type	Direction	Description
Key_id	Key_ID	In	Identität des angefragten Schlüssels
Key	Cryptographic_Key	Out	Der Wert des geladenen Schlüssels
	ReturnCode	return	„Key_load_completed“: Das Laden des Schlüssels war erfolgreich und der in „Key“ zurückgegebene Wert enthält die Schlüsseldaten. „Key_not_found“: Es konnte kein Schlüssel mit der in Key_id angegebenen Identifikation gefunden werden. „Key_load_unauthorized“: Der Schlüssel befindet sich auf der zentralen KD_ECU und die anfragende ECU hat nicht die für den Zugriff nötige Autorisation. „Key_get_key_unauthorized“: Der Schlüssel darf das Security Framework nicht verlassen.

7.5.2 Funktion: get_key_handle()

- Funktionsbeschreibung: Diese Methode erlaubt indirekten Zugriff auf kryptographische Schlüssel. Sie kann von jedem Modul oder jeder Applikation aufgerufen werden, welche Schlüssel verwenden. Diese Funktion führt transparent die Schritte analog zu „get_key()“ aus, gibt jedoch nicht den eigentlichen Schlüssel heraus, sondern einen Handle auf den geladenen Schlüssel. Dieser Handle kann im Krypto-Modul für kryptographische Operationen verwendet werden. Mit dieser Methode wird der kryptographische Schlüssel nicht aus dem Security Framework herausgegeben und dessen Sicherheit ist gewährleistet.
- Input/Output-Format:

Parameter	Type	Direction	Description
Key_id	Key_ID	In	Identität des angefragten Schlüssels
Handle_id	Handle_ID	Out	Der Handle auf den geladenen Schlüssel
	ReturnCode	return	„Key_load_completed“: Das Laden des Schlüssels war erfolgreich und der in „Key“ zurückgegebene Wert enthält die Schlüsseldaten. „Key_not_found“: Es konnte kein Schlüssel mit der in Key_id angegebenen Identifikation gefunden werden. „Key_load_unauthorized“: Der Schlüssel befindet sich auf der zentralen KD_ECU und die anfragende ECU hat nicht die für den Zugriff nötige Autorisation.

7.5.3 Funktion: get_key_properties()

- Funktionsbeschreibung: Diese Methode gibt Zugriff auf die Eigenschaften eines kryptographischen Schlüssels. Eigenschaften sind hier Informationen über das verwendete kryptographische Verfahren, Lebenszeit, evtl. Autorisation, etc..
  Analog zu get_key() holt sich diese Funktion die Informationen lokal oder von einer KD-ECU.
- Input/Output-Format:

Parameter	Type	Direction	Description
Handle_id	Handle_ID	In	Handle auf den Schlüssel dessen Eigenschaften angefragt werden

Parameter	Type	Direction	Description
Properties	Property_Array	Out	Ein Array, das eine Tabelle aus Property-Namen und Wert beinhaltet
	ReturnCode	return	„Key_get_properties_completed“: Das Laden der Eigenschaften des Schlüssels war erfolgreich und die in „Propertes“ zurückgegebene Wert enthält eine Liste der Eigenschaften. „Key_not_found“: Es konnte kein Schlüssel mit der in Key_id angegebenen Identifikation gefunden werden. „Key_get_properties_unauthorized“: Die anfragende ECU hat nicht die für die Anfrage nötige Autorisation.

7.5.4 Funktion: insert_key()

- Funktionsbeschreibung: Mit dieser Methode werden Schlüssel zur Verwaltung im Key Management abgelegt. Diese Methode dient vor allem dem initialen Befüllen des Key Managements durch Importieren, aber auch zu Wartungszwecken. Diese Methode führt intern eine Überprüfung der Berechtigung der anfragenden Instanz beim Policy Management durch.
- Input/Output-Format:

Parameter	Type	Direction	Description
Key_id	Key_ID	In	Identität des einzufügenden Schlüssels. Mit dieser Identität kann der Schlüssel später wieder abgefragt werden.
Key	Cryptographic_Key	In	Kryptographischer Schlüssel, der eingefügt werden soll
Properties	Property_Array	In	Ein Array, das eine Tabelle aus Property-Namen und Wert beinhaltet
IsToBeStoredLocally	boolean	In	Legt fest, ob der Schlüssel lokal oder auf der KD_ECU gespeichert werden soll. Falls der Schlüssel auf der KD_ECU gespeichert wird, wird dieser an die KD_ECU gesendet.
SecData	Security_Metadata	In	Identifikation des Benutzers/Prozesses, der diese Aktion verlangt. Diese Daten werden intern für einen Policy-Check verwendet.
	ReturnCode	return	„Key_insertion_completed“: Das Einfügen des Schlüssels war erfolgreich. „Key_ID_invalid“: Die angegebene Identität des neuen Schlüssels ist ungültig oder bereits einem anderen Schlüssel zugewiesen. „Key_insertion_unauthorized“: Das Einfügen des Schlüssels wurde aufgrund von Policing abgelehnt.

7.5.5 Funktion: create_key()

- Funktionsbeschreibung: Mit dieser Methode werden Schlüssel im Key Management erzeugt. Diese Methode dient dem Erzeugen von Schlüsseln, die nicht von anderen Quellen importiert werden sollen. Diese Methode führt intern eine Überprüfung der Berechtigung der anfragenden Instanz beim Policy Management durch.

- Input/Output-Format:

Parameter	Type	Direction	Description
Key_id	Key_ID	In	Identität des einzufügenden Schlüssels. Mit dieser Identität kann der Schlüssel später wieder abgefragt werden.
Properties	Property_Array	In	Ein Array, das eine Tabelle aus Property-Namen und Wert beinhaltet
IsToBeStoredLocally	boolean	In	Legt fest, ob der Schlüssel lokal oder auf der KD_ECU gespeichert werden soll. Falls der Schlüssel auf der KD_ECU gespeichert wird, wird dieser an die KD_ECU gesendet.
SecData	Security_Metadata	In	Identifikation des Benutzers/Prozesses, der diese Aktion verlangt. Diese Daten werden intern für einen Policy-Check verwendet.
	ReturnCode	return	„Key_insertion_completed“: Das Einfügen des Schlüssels war erfolgreich. „Key_ID_invalid“: Die angegebene Identität des neuen Schlüssels ist ungültig oder bereits einem anderen Schlüssel zugewiesen. „Key_insertion_unauthorized“: Das Einfügen des Schlüssels wurde aufgrund von Policing abgelehnt.

7.5.6 Funktion: delete_key()

- Funktionsbeschreibung: Mit dieser Methode werden Schlüssel aus Verwaltung im Key Management gelöscht. Diese Methode dient Wartungszwecken. Sie führt intern eine Überprüfung der Berechtigung der anfragenden Instanz beim Policy Management durch.
- Input/Output-Format:

Parameter	Type	Direction	Description
Key_id	Key_ID	In	Identität des zu löschenden Schlüssels.
SecData	Security_Metadata	In	Identifikation des Benutzers/Prozesses, der diese Aktion verlangt. Diese Daten werden intern für einen Policy-Check verwendet.
	ReturnCode	return	„Key_deletion_completed“: Das Löschen des Schlüssels war erfolgreich. „Key_not_found“: Es konnte kein Schlüssel mit der in Key_id angegebenen Identifikation gefunden werden. „Key_deletion_unauthorized“: Das Löschen des Schlüssels wurde aufgrund von Policing abgelehnt.

7.5.7 Funktion: revoke_key()

- Funktionsbeschreibung: Mit dieser Methode werden Schlüssel unter Verwaltung des Key Management als ungültig markiert ohne sie zu löschen. Diese Methode dient Wartungszwecken. Sie

führt intern eine Überprüfung der Berechtigung der anfragenden Instanz beim Policy Management durch.

- Input/Output-Format:

Parameter	Type	Direction	Description
Key_id	Key_ID	In	Identität des zu revozierenden Schlüssels.
Key	Cryptographic_Key	In	Kryptographischer Schlüssel, der eingefügt werden soll
SecData	Security_Metadata	In	Identifikation des Benutzers/Prozesses, der diese Aktion verlangt. Diese Daten werden intern für einen Policy-Check verwendet.
	ReturnCode	return	„Key_revocation_completed“: Das Löschen des Schlüssels war erfolgreich. „Key_not_found“: Es konnte kein Schlüssel mit der in Key_id angegebenen Identifikation gefunden werden. „Key_revocation_unauthorized“: Das Löschen des Schlüssels wurde aufgrund von Policing abgelehnt.

7.6 ATTESTATION ZUR ABSICHERUNG DER SCHLÜSSELSPEICHER

In den oben beschriebenen Lösungsansätzen gibt es mehrere Angriffspunkte. Zum einen sind im lokalen Flashspeicher hinterlegte Schlüssel grundsätzlich nicht gegen unbefugten Zugriff geschützt, zum anderen fehlt beim Szenario der zentralen Schlüsselspeicherung eine Vertrauensbeziehung zwischen Client-ECU und KD-ECU. Es findet zwar eine Authentifizierung der Kommunikation zwischen den beiden ECUs statt, jedoch kann aufgrund einer validen Authentifizierung kein Rückschluss auf die Unkompromittiertheit eines Systems geschlossen werden. Die KD-ECU ist zwar mit einem sicheren Schlüsselspeicher ausgestattet, ein Angreifer kann aber versuchen die restlichen Komponenten im Flash-Speicher (wie z.B. das Betriebssystem und die installierten Anwendungen) zu manipulieren. Eine Client-ECU kann ein Angreifer theoretisch beliebig manipulieren und alle gespeicherten Daten wie kryptographische Schlüssel auslesen. Es sind Mechanismen notwendig, solche Manipulationen zu erkennen bzw. Methoden wie ECUs gegenüber entferneten Kommunikationspartnern nachweisen können, dass keine Manipulationen stattgefunden haben. Hierzu können Attestationsprotokolle eingesetzt werden. Im Folgenden werden allgemeine Verfahren beschrieben, wie sich diese beiden ECUs gegenseitig attestieren können.

7.6.1 Attestation der KD-ECU gegenüber der Client-ECU

Bei dieser Attestation weist die KD-ECU gegenüber den Client-ECUs nach, dass sie nicht kompromittiert ist. Das hier vorgeschlagene Verfahren basiert auf der Anwendung der in [TODO Ref auf Plattform-Integritäts-Test] beschriebenen Anwendung von Plattform-Integritäts-Tests basierend auf einem HSM.

Beim Aufstart der KD-ECU wird vom HSM ein Plattform-Integritäts-Test durchgeführt, um zu prüfen, ob die Firmware der KD-ECU kompromittiert wurde. Bei erfolgreicher Absolvierung wird der Zugriff auf die hinterlegten Werte im sicheren Speicherbereich des HSMs freigeschaltet. Neben den gespeicherten Schlüsseln für die Client-ECUs sind ebenfalls Werte zur Attestation der KD-ECU gegenüber den Client-ECUs sicher gespeichert. Diese Werte werden in Attestationsprotokollen eingesetzt und attestieren die Unkompromittiertheit der KD-ECU. Die Client-ECUs vergleichen die empfangenen Werte mit ihnen bekannten Refrenzwerten und können darauf schließen, dass das HSM der KD-ECU den Plattform-Integritäts-Test erfolgreich durchgeführt hat und die KD-ECU als unkompromittiert erachtet. Diese Refrenzwerte sind derart zu gestalten, dass sie öffentlich bekannt sein können und trotzdem die Validität der Attestation nicht gefährden. Einen möglichen Lösungsansatz für dieses Problem bieten Hashketten.

Das für die Attestation verwendete Protokoll kann u. a. auf einem Challenge-Response-Verfahren oder Hash-Chains beruhen.

Es ist jedoch zu berücksichtigen, dass diese Kommunikation authentisch sein muss. Die hierfür nötigen Daten sind entweder in der Client-ECU unsicher im Flash-Speicher hinterlegt oder können (falls für die Authentifizierung Eigenschaften wie Laufzeitverhalten verwendet werden) simuliert werden. Hieraus ergibt sich grundsätzlich das Problem der Man-In-The-Middle Angreifbarkeit. Dieses kann nur gelöst werden indem auf der Client-ECU ein sicherer Speicher verbaut wird.

7.6.2 Attestation der Client-ECUs gegenüber der KD-ECU

Wie oben beschrieben, stellt das Speichern von Schlüsseln im Flash-Speicher ein grundsätzliches Sicherheitsproblem im Sinne von auslesbaren oder veränderbaren Schlüsseln dar. Die Attestation der Client-ECU gegenüber der KD-ECU kann dieses Problem zwar nicht beseitigen, bietet jedoch eine Möglichkeit eine bereits manipulierte ECU zu erkennen und dementsprechend Reaktionen einzuleiten.

Die hier verwendbaren Verfahren können auf in Kapitel 4.5 beschriebenen Hardware- und in [14] Kapitel 3.5 beschriebenen Software-basierten Attestationsverfahren beruhen.

8. POLICY MANAGEMENT-MODUL

Policy Management ist ein zentrales Problem für System- und Anwendungssicherheit. Es geht darum, Zugriffskontrollrichtlinien zu definieren und die Ressourcen mit ihren entsprechenden Zugriffs- und Kontrollrechten zu charakterisieren. In der Praxis bedeutet es oft, eine Liste aller erlaubten Status zu erstellen.

Authorization und Access-Control Design bauen auf einigen Prinzipen auf [4]:

- *„Principle of least privilege“:* eine Entität soll nur die minimal nötigen Rechte bekommen, um sein Ziel zu erreichen;
- *„Principle of fail-safe defaults“:* Zugriff auf Objekte wird grundsätzlich verweigert, außer es liegt eine explizite Berechtigung der anfragenden Entität vor;
- *„Principle of economy of mechanism“:* Sicherheitsmechanismen müssen so einfach wie möglich sein;
- *„Principle of complete mediation“:* Alle Zugriffe auf Objekte müssen jedes Mal vom Verantwortlicher der Resource bestätigt werden, um zu vermeiden, irrtümlich eine falsche Erlaubnis zu geben;
- *„Principle of open design“:* Die Sicherheit eines Mechanismus darf nicht von der Geheimhaltung seiner Implementierung abhängen (Kerkhoff‘sches Prinzip [Referenz]);
- *„Principle of separation of privileges“:* Ein System soll nicht basierend auf einer einzelnen Sicherheitsbedingung eine Erlaubnis erteilen, nur eine Bedienungbestätigung kann nicht genug für eine starke sivherheit sein (z.B. 2 Checks are besser.);
- *„Principle of least common mechanism“:* Mechanismen, die Zugriff auf einem Objekt geben, dürfen nicht zwischen 2 unterschiedlichen Prozessen geteilt sein;
- *„Principle of psychological acceptability“:* Wegen Sicherheitsmechanismen sollte das Objekt nicht schwieriger zu erreichen und zuzugreifen sein als ohne.

8.1 ACCESS CONTROL

Es gibt mehrere Ansätze für Techniken zur Zugriffskontrolle:

- *„Discretionary Access Control“ (DAC) [12]:* DAC erlaubt dem Besitzer der Informationen zu entscheiden, wer ein bestimmtes Objekt lesen, modifizieren (schreiben) und ausführen kann. DAC basiert auf der Idee, dass der Eigentümer der Daten bestimmen soll, wer Zugang zu ihr hat.
- *„Mandatory Access Control“ (MAC) [3]:* MAC ist eine Technik, in der der Zugang vom System und nicht vom Eigentümer der Daten bestimmt wird. Administratoren und Überwachungsbehörden bestimmen im Voraus, wer auf ein Objekt zugreifen darf und wer nicht.
- *„Role-Access based Control“ (RBAC) [29]:* RBAC verlangt, dass Zugriffsrechte auf Rollen statt auf einzelne Subjekte zugeordnet werden. Die Subjekte erhalten diese Zugriffsrechte durch zugewiesene Mitgliedschaft in entsprechenden Rollen.
- *„Domain Type Enforcement“ [2]:* jedem User wird eine Domäne und jeder Ressource einen Typ zugeordnet, eine Manipulation von Resourcen vom einem spezifischen Typ wird nur für eine Gruppe von spezifischen Domänen (dh Gruppe von User-Grupen) erlaubt;

Die Implementierung solcher Policies folgt zwei Ansätze bzw. Securitymodellen:

- *“Acces Control List based”* (ACL) [29]: ACL ist eine Liste von Entitäten, die Zugriff zu einem Objekt haben. ACL ist ein Konzept der Trennung von Privilegien, in dem jedes Objekt mit eine Menge von Tupeln/Paaren (Subjekt und Set von Rechten) verbunden ist. Die Liste ist eine Datenstruktur (z.B. eine Tabelle), deren Einträge die Rechte jedes Subjektes für ein bestimmtes Objekt definieren.

- "*Acces Control Capability based*" [29]: Ein Capability ist eine fälschungssichere Eigenschaft oder ein Token (Challenge, Passwort, kryptographischer Schlüssel…) und ihr nachgewiesener Besitz bietet Zugriff zu einer Ressource;

Viele Sprachen/Protokolle wurden entwickelt, um Policies zu beschreiben:

- XACML [22] ist ein XML-Schema, das die Darstellung und Verarbeitung von Autorisierungs-Policies standardisiert.
- SAML [19] ist ein XML-Framework zum Austausch von Authentifizierungs- und Autorisierungsinformationen. Es erlaubt Authentication Statements (Zusicherung einer Authentifizierung), Attributes Statements (Zusicherung der Verfügbarkeit eines Objektes) und Authorization Decision Statements (Autorisierung für eine bestimmte Ressource).

Im Projekt EVITA fiel die Entscheidung für eine Implementierung eines Role-based Control Access mit einer Access Control List (ACL) in XACML.

8.2 POLICY- UND AUTORISIERUNGSMANAGEMENT

8.2.1 Policy Definition

Das Policy Modul ist verantwortlich für die Applikations- und Middlewarerechte, es muss einfach und sicher Zugriff (bzw. Kenntnis) auf diese Rechte haben und in der Lage sein, den Zugriff zu einer Ressource wirkungsvoll zu kontrollieren. Diese Applikations- und MW-rechte betreffen den Zugriff zu einer Ressource, und definieren [9]:

- Die erlaubten Informationen und Kontext für die Applikation;
- Das Ressourcen-Management (Verbindung, Bandbreite, CPU-Kapazität…);
- Die erforderten Security-Mechanismen (Protokolle, Security Credentials…);

Sie bestehen aus:

- Einem Subjekt: Wer (Dienst, User, Sensor, ECU) erfragt Zugriff zu einer Ressource;
- Einem Target: die Ressource (Dienst, User, Sensor, ECU, Datenbank) selbst und ihre Beschreibung;
- Einer Policy-Aussage: das Ergebnis für die Rechtsnachfrage (erlaubt, verboten, durchsetzen);
- Weiteren Attributen: andere Attributen können angefügt werden, um den Kontext genauer anzugeben (Bedingung für den Prozess, Zeit, Ort, Status des Systems/Fahrzeugs, MW-sequence events…), diese Attribute machen die Zugriffskontrolle granularer;

In den nächsten Tabellen (Tabelle 12) sind weitere Kriterien für die Policy-Beschreibung vorgestellt:

Identitätskriterien (für Quelle- und Zielsteuergerät)	Beispiele
Entitätsidentität	ECU-Name, ServiceID, IP-Adresse, Port
Klasse des Steuergerätes	ECU, Sensor, Actuator, CE-Device, Backend Server…
Rolle	RPC-server, RPC-client, Pub-Server, Client-Subscriber, SD-Directory, SD-provider, SD-user, interne, externe Kommunikation, Sicherheitsklasse, Security-Modul
Umgebungspezifizierung	Version

Identitätskriterien (für Quelle- und Zielsteuergerät)	Beispiele
Security-Spezifizierung	Sicherheitszone und CIs, Domäne, verbundene Schlüssel, Connection-Handle

Tabelle 11: Policy-Identitätskriterien

Manipulationsziele	Beispiele
Allgemeines Ziel	lesen, schreiben, durchführen, weiterleiten, anschließen, kommunizieren, überprüfen, verschlüsseln, signieren, Filter addieren…
Objektbeschreibung (erwartet, gesendet, modifiziert)	Objektsklasse (String, Integer, JavaScript-Code, Key-Handle, Communication-Handle, SecurityObjekt, Security Check …); Identität des Objektes (wenn spezifisch für das gesamte System, Geschwindigkeit, Position des Autos, Verfügbarkeit von Security Plug-ins, Signatur von X, HMAC für Y,…)

Tabelle 12: Policy-Aktionsbeschreibung

Jede Policy ist eine Mischung dieser Attributen und ihre Anwesenheit und Präzision definieren die Granularität der Policy. Die Granularität der Policiydefinition hängt vom jeweiligen Use-Case und vom OEM ab.

Praktisch werden drei Arten von Policies existieren:

- Die Entwicklungs-Policy: diese Ebene betrifft die Entwicklungsphase und die manuelle Einführung von Policies. Die Entwickler können ihre eigenen Policies auf einer abstrakten Sicherheitsskala in einer lesbaren Form definieren. Zum Beispiel: 1) für die Kommunikation: "*X* can communicate with *Y* over a *very secure* channel and can exchange only *object O*", hier charakterisiert "very secure" das Security-Level der Verbindung abstrakt; 2) für Objekt-Zugriffskontrolle: "The resource A can be accessed only by an entity from domain D and the security zone Z in the context C ". Diese Sorten von Policies werden nur für die Einführung von neuen Policies (z. B. in XACML, SAML) verwendet und werden sofort in einem maschinlesbaren Format gespeichert oder in ein neues Format übersetzt. Diese neugenerierten Policies werden von MW und Applikation verwendet. Die Policy kann die Sicherheitsanforderungen in einer abstrahierten Weise wie in (1) und (2) oder mit direkten Sicherheitsspezifizierungen (zB TLS, IPSec) angeben.
- Eine Application-level Policy definiert die Anwendungsrechte, sie kann direkt von der Anwendung oder dem SCM konsultiert werden. Sie verfolgt das Ziel, eine feinere Zugriffskontrolle und eine direkte Durchsetzung von Sicherheitsrichtlinien im Applikations-Code zu bieten, zum Beispiel: " the application A, for the N^{th} step, can only use the type of object O". Mit solchen Policy kann das PMM die Eigenschaften des betroffenen Objektes testen, einen Filter setzen, die Objektherkunft überprüfen, den Prozess beenden oder den Zugriff auf eine Ressource verweigern. Diese Sicherheits-Prozesse können in der MW stattfinden. Die SCM können solche Policies heranziehen, um einen sicheren Kanal einzurichten; die Grundidee ist, den erlaubten Kontext der Kommunikation klar zu evaluieren. Diese Policies können aus den Development-level Policies abgeleitet werden.
- Eine Middleware-level Policy: mit diesen Policies kann die MW die Konfiguration der Kommunikation oder der Datenverarbeitung setzen. Jede Komponente des Security-Frameworks muss bestimmte Sicherheits-Policies vom PMM befolgen.

  Das Initiator-SCM (im Verbindungsprozess) fragt zuerst nach der Kommunikations-Policy aus dem geeigneten PMM und bekommt die Policy-Decision für das Protokolle-plug-in (z.B. ID des Conneciton-Handle, gültiger Eintrag in der IPSec-Policy-Datenbank), für den Schlüssel (e.g. Aus-

tausch-Protokolle, ID vom Key-Repository) und für andere Sicherheitsmaßnahmen (z. B. für die Überprüfung der digitalen Signaturen oder HMACs in der HSM).

Gleichzeitig informiert das PMM das Receiving-SCM über den Kommunikationsinitiator und vermittelt die gültige Policy-Decision (e.g. EntityID, Protokolle, Connection-Handle, Schlüssel, Daten-Filter…).

Das PMM muss die anderen Security-Module auch über die aktuell laufenden Prozesse informieren. Dazu sind zwei Lösungen möglich: Entweder bekommt das SCM die Liste von Policies für alle anderen Security-Module (KMM, CSM…) und ist für die Weiterleitung verantwortlich, oder jedes Sicherheitsmodul bekommt direkt vom PMM seine eigenen Policies. Die erste Lösung ist exklusiv auf das fehlerfreie und zuverlässige Funktionieren des SCM angewiesen, erzeugt jedoch weniger Netzwerkverkehr.

8.2.2 Autorisierungskomponenten

Ein Policy Modul besteht mindestens aus 2 Entitäten: dem „Policy Enforcement Point“ (PEP) und dem „Policy Decision Point“ (PDP).

8.2.2.1 PDP

Der Policy-Decision-Point (PDP) ist die Entität, die die Zugriffskontroll-Funktionalität für das System anbietet. Er hat Zugriff zu einer lokalen oder verteilten Datenbank von Policies. Der Policy-Enforcement-Point (PEP) erfragt bei ihm, die jeweiligen Zugriffsrechte eines Prozesses auf eine Ressource.

- Autonomer PDP: Normalerweise agiert der PDP autonom, das heißt er trifft die Entscheidungen (Policy-Entscheidungen) für alle Policyanfragen zu seiner Domäne;
- Verteilter PDP: Der PDP kann Entscheidungen auch zu einem anderen PDP delegieren. Für eine solche Interaktion wird Funktionalität zum Aufbau und zur Bestätigung von Vertrauensbeziehungen benötigt (Sichere Kanäle-Modul);

8.2.2.2 PEP

Der Policy-Enforcement-Point (PEP) ist ein weiteres Submodul des Autorisierungsmoduls. Er bekommt die Anfrage einer Entität zum Zugriff auf eine Ressource, lässt die entsprechende Policy Entscheidung vom PDP durchführen. gewährt entsprechend Zugriff zur Ressource oder nicht. Ein PEP kann auch in vielen anderen Sicherheitsmodulen oder Programmen integriert sein, die Sicherheitsressourcen kontrollieren (Authentifizierung, Key Management…).

- Autonomer PEP: Der PEP agiert wie ein unabhängiges Autorisierungsmodul, er kann seine eigene Policy Entscheidung lokal speichern;
- Weiterleitender PEP: Der PEP leitet die Policy-Entscheidungsanfrage zu einem PDP und anschließend die Antwort des PDP an das Subjekt weiter. Dies ist die standardmäßige Implementierung, die zeitliche und kontextabhängige (Latenzzeit, Network Overhead…) Beschränkungen mit sich bringen kann.

Für die Fälle in denen der Weiterleitende PEP (i) Sicherheitskontext-Information hat, die für andere PDPs nicht verfügbar oder nicht verständlich sind, oder (ii) eine zeitkritische Entscheidung treffen muss, muss der PEP seine Entscheidung auf seiner eigenen Policy-Datenbank basierend treffen.

Die meisten Sicherheitsmodule sind von Natur aus Automer PEPs aus Performanzgründen.

8.2.2.3 PEP/PDP Struktur

In unserem System gehen wir davon aus, dass sowohl der Empfänger als auch der Sender die Policy für jede Nachricht auswerten.

Entscheidungsprozess für eine eingehende Nachricht

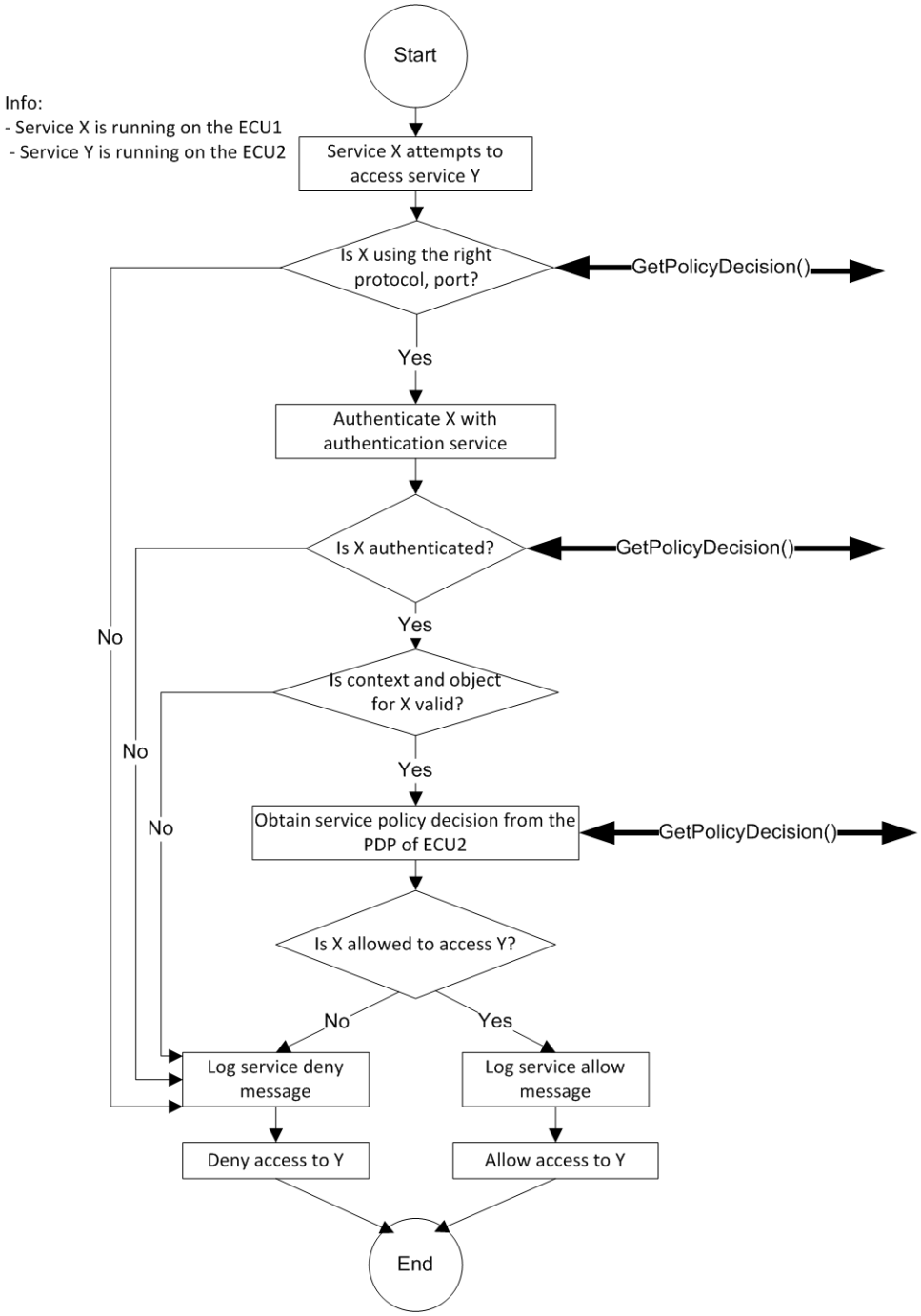

Abbildung 14: State Machine des PEPs für eine eingehende Nachricht

In Abbildung 14 wird der Entscheidungsprozess auf dem PEP für eine eingehende Nachricht dargestellt. Standardmäßig werden alle Operationen abgelehnt, außer es existieren für die jeweilige Operation eine gültige Filter-Policy, eine gültige Authentifizierung-Policy und eine gültige Kontext-Policy.

Entscheidungsprozess für eine ausgehende Nachricht

In Abbildung 15 wird der Entscheidungsprozess auf dem PEP für eine eingehende Nachricht dargestellt. In diesem Fall betrachten wir eine statische Konfiguration der Policy, das heißt, zur Laufzeit kann und darf kein Prozess eine Policy im System (das heißt in der Datenbank) hinzufügen oder verändern. In diesem

Prozess bestätigt wird schon vor der Vermittlung der Nachricht anhand der Policy-Datenbank überprüft, ob der Empfänger das Paket akzeptieren wird.

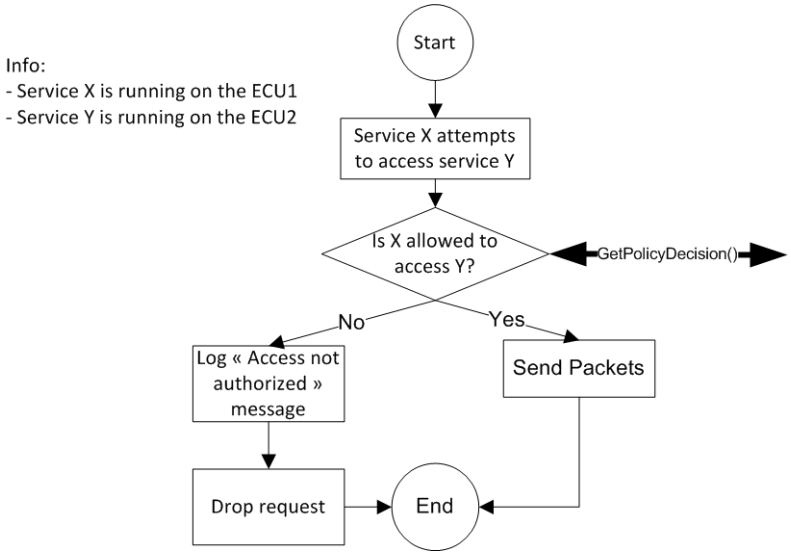

Abbildung 15: State Machine des PEPs für eine ausgehende Nachricht mit statischer Policy-Konfiguration

Zugriff auf Policy-Entscheidungen

Für jede sicherheitsrelevante Entscheidung überprüft der PEP, ob die Policy-Entscheidung lokal gespeichert ist. Falls dies nicht der Fall ist, erfragt er die Policy-Entscheidung beim geeignetsten PDP (siehe Abbildung 16 und Abbildung 17).

Auf der anderen Seite bearbeitet der PDP die Anfrage und sucht eine entsprechende Policy zuerst lokal und dann ggf. unter Zuhilfenahme anderer PDPs, um zu einer Policy-Entscheidung für den entsprechenden Fall zu kommen. Wenn kein Policy gefunden wird, ist die Policy-Entscheidung „Autorisierung Abgebrochen“.

Die Freshness von Policy-Entscheidungen muss abgesichert werden, da die Policy-Datenbank aktualisiert werden kann. Deshalb braucht das Autorisierungsmodul ein System, um die Entscheidungen, die lokal gespeichert sind, aktuell zu halten (im Push-Modus oder Pull-Modus).

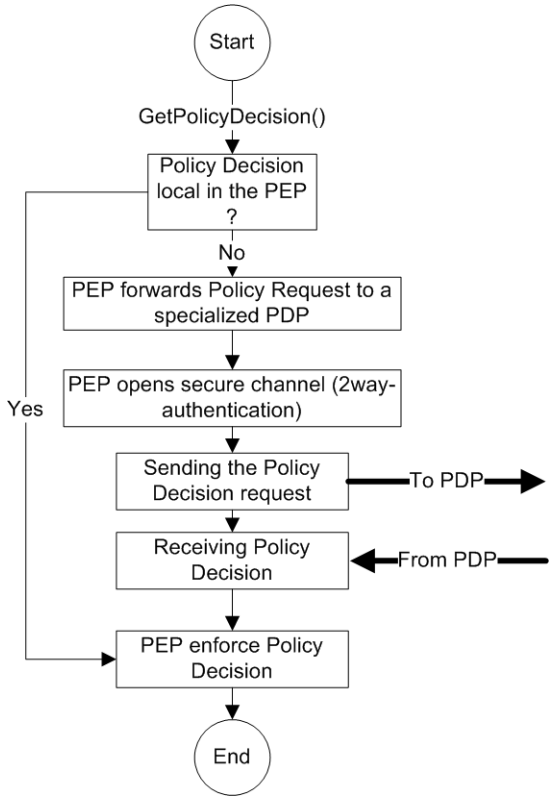

Abbildung 16: GetPolicyDecision() (PEP-Seite)

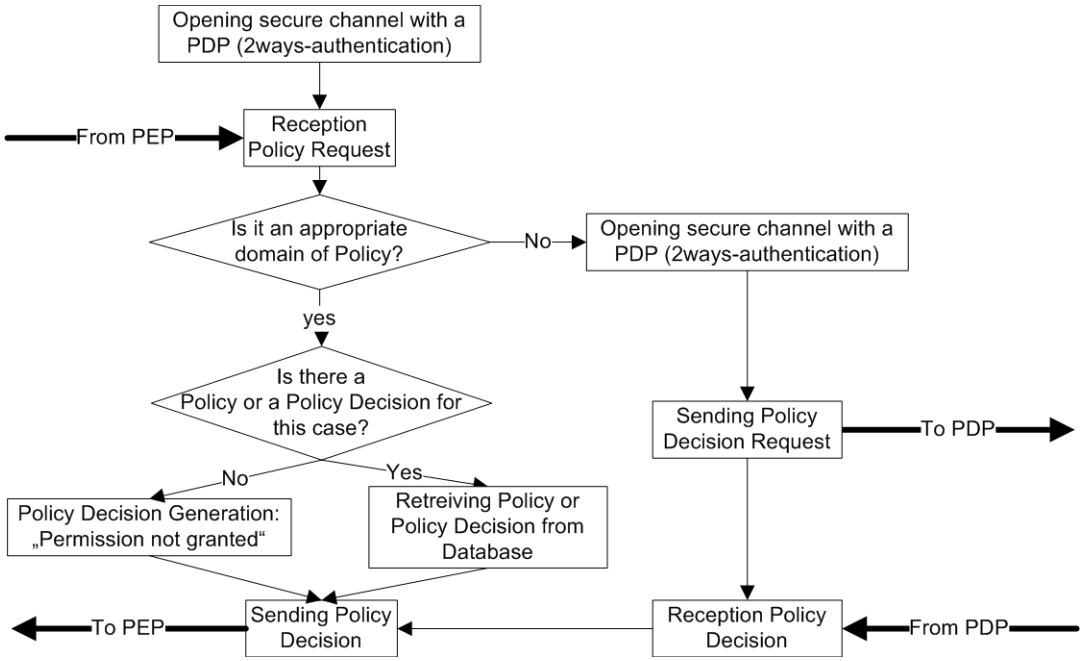

Abbildung 17: GetPolicyDecision() (PDP-Seite)

Policies und Policy-Entscheidungsspeicherung

Policies müssen in einem sicheren Speicher gespeichert werden:

- Nur erlaubte und authentifizierte Entitäten dürfen Zugriff (lesend und schreibend) zur Datenbank und nur über einen sicheren Kanal haben;
- Die Datenbank muss verschlüsselt sein, und der Verschlüsselungsschlüssel muss in einem sicheren Hardware-Module gespeichert sein.

Policy-Entscheidungen werden von den PDPs generiert und sowohl PEP als auch PDP können sie speichern. Auch die Entscheidungen müssen sicher gespeichert werden:

- Nur erlaubte und authentifizierte Prozesse dürfen Zugriff (Lesen, Schreiben und Austauschen zwischen PDP und PEP) zur Datenbank für Policy-Entscheidungen und nur über einen sicheren Kanal haben;

8.2.3 Security Policy Installation

Policies können vom OEM, dem Entwickler und der Middelware selbst editiert warden (bspw. falls die Kommunikation zwischen zwei Zonen über IPsec erfolgen muss, erstellt die Middleware (PMM) für jede Dienstekommunikation eine Policy „Service A kommuniziert mit Service B über IPsec“.). Die Installation von Policies erfolgt zur Laufzeit, am Bandende oder während eines Softwareupdates durch den OEM.

8.2.3.1 Authorization Ticket

Ein Authorization-Ticket ist eine Datenstruktur, die eine Application-level Policy beinhaltet. Die Integrität und die Authentizität des Tickets müssen überprüfbar sein (z.B. durch eine digitale Signatur) und aus einer autorisierten Quelle (z.B. OEM) stammen.

8.2.3.2 Security Configuration Ticket

Ein Security Configuration Ticket einer MW-level Policy ist ein Set von Regeln (z.B. Firewall-Regeln, Virus-Signatur Datenbank, Regeln zur Festlegung der zu benutzenden Sicherheitsmechanismen) um einen PEP/PDP oder eines seiner Untermodule zu konfigurieren. Wie ein Authorization Ticket müssen die Integrität und die Authentizität überprüfbar sein und aus einer autorisierten Quelle stammen.

8.2.3.3 Installationsprozess

Das System installiert neue Policies auf zwei Wegen (siehe Abbildung 18):

- Für das Authorizationsticket: Der PDP überprüft die Ursprungsauthentisierung, bestätigt, wenn die Installation autorisiert ist, und endlich führt den Installationsprozess in der geigneten Policy-Datenbank durch.
- Für das Security Configuration Ticket: Der PDP überprüft die Ursprungsauthentisierung, bestätigt, wenn die Installation autorisiert ist, und am Ende schickt die neue Konfiguration zum richtigen Komponenten, damit er sein Konfiguration (z.B. Set von Filtern) aktualisiert kann.

Die Security Plug-Ins des Autorisierungsmoduls sichern die Ticket-Übermittlung aus dem entsprechenden Speicher zum dem PDP/PEP und die Übersetzung in die jeweilige PDP/PEP-Sprache.

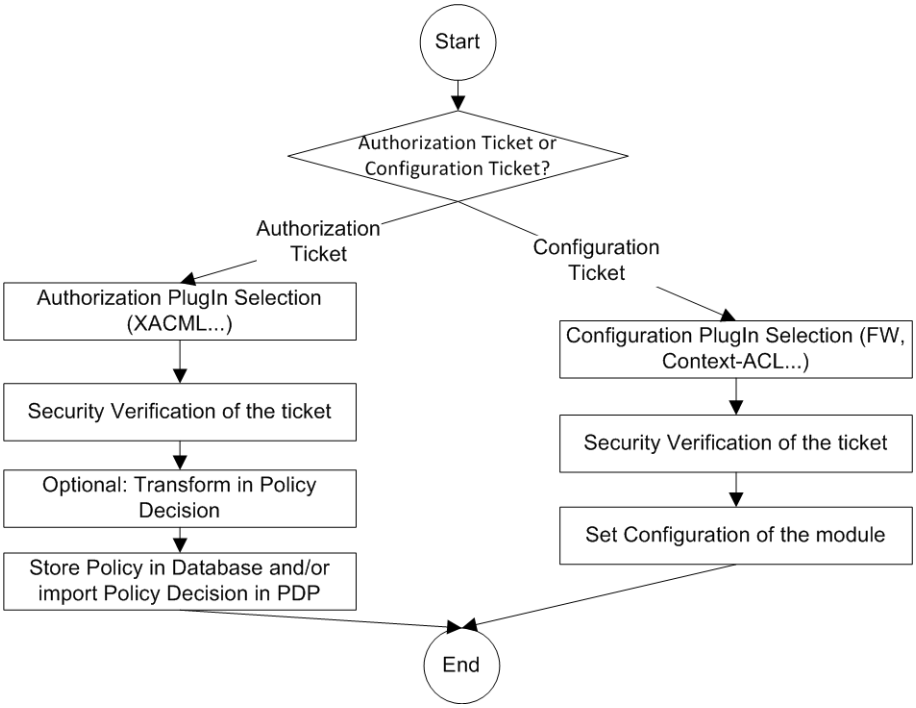

Abbildung 18: Autorisierungsinstallationsprozess

8.2.4 Autorisierung und externe Kommunikation

Innerhalb des Fahrzeugs ist die Einbringung weiterer interner Geräte während der Laufzeit unwahrscheinlich. Alle Policies, die die interne Kommunikation betreffen, können am Bandende oder während der Installation eines OEM-Updates konfiguriert werden. Zur Laufzeit sind solche Policies statisch.

Auf einer anderen Seite zwingt die Integration eines externen Geräts das Fahrzeug, neue geeignete Policies hinzuzufügen. Die Konfiguration dieser Policies wird dynamisch zur Laufzeit verhandelt.

Für externe Kommunikation spielt der Proxy des Fahrzeugs die Rolle eines Mediators. Das externe Gerät wendet sich an ihn, um Zugriff zum internen Netzwerk zu erhalten. Der Proxy verhandelt das Session-Management für diese Geräte, und erstellt die benötigten Policies für sie bzw. für die Kommunikations mit ihnen.

Nach dieser Anmeldung muss der Proxy die autorisierten Nachrichten vom und zum externen Gerät weiterleiten, die Kommunikation zwischen Proxy und internen Diensten ist schon statisch konfiguriert. Also verbindet der Proxy die dynamische Anmeldung externer Geräte mit dem bestehenden Set statischer Policies.

Der Proxy klassifiziert jedes Gerät in einer Sicherheitsklasse, die dann die Sicherheitsanforderungen und das Vertrauen in die Verbindung zwischen diesen beiden Entitäten definiert. Eingehende Nachrichten leitet der Proxy nach einer Sicherheitsanalyse (basiert auf statischen Policies) weiter, und fügt ggf. weitere Attributen an, um die Verbindung, das Außengerät und die Nachrichtenanalyse zu charakterisieren. Für ausgehende Nachrichten übermittelt der interne Dienst zum Proxy die Nachricht und weitere Attribute, die die Sicherheitsanforderungen des Dienstes an die externe Kommunikation definieren.

8.3 MODULARCHITEKTUR

In der Abbildung 19 wird der Architektur Vorschlag für das AMM vorgestellt. Der PEP wird auf mehreren Schichten des ISO/OSI-Modelles eingebaut, und besteht aus 3 Schichten an Granularität. Die zwei Untermodule des Security Policy Moduls haben ihre eigenen Policy-Formate, basierend auf den Policy Definitionen des vorliegenden Dokuments.

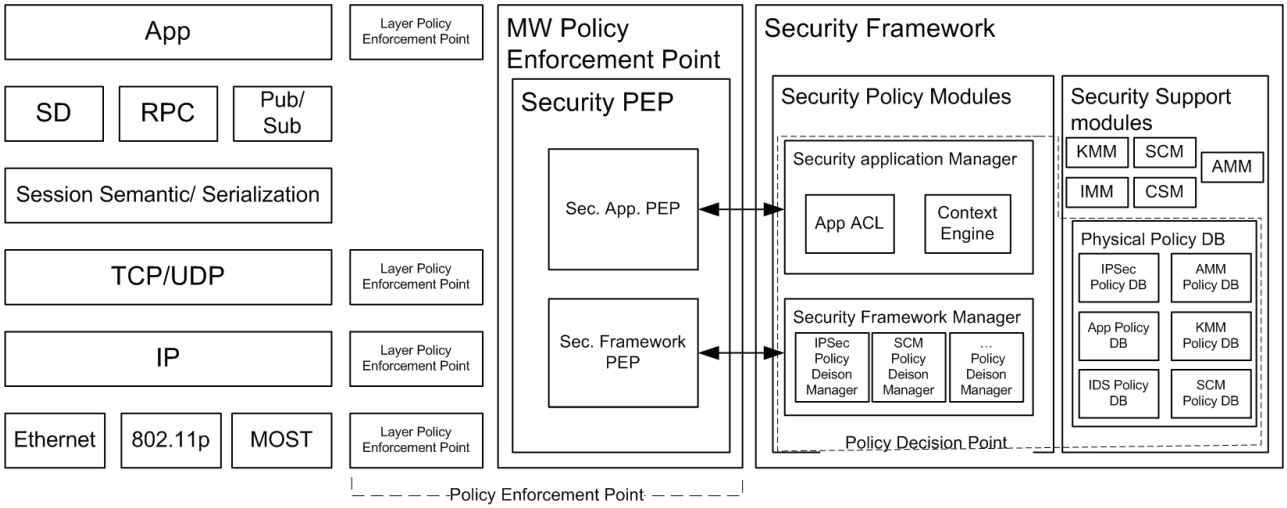

Abbildung 19: AMM (PEP+PDP) & MW-Architektur

8.3.1 Struktur des PDPs

Der PDP besteht aus zwei Hauptkomponenten: dem Security Application Manager und dem Security Framework Manager. Diese zwei Komponenten werden von anderen Sicherheitsmodulen unterschtützt, z.B CSM und KMM für die Policy Speicherung, oder AMM , SCM, CSM und KMM für die Kommunikation zwischen PDP und PEP (wenn nicht auf der gleichen ECU implementiert). Der Security Application Manager verwendet Application-level Policies und der Security-Framework Manager verwendet MW-level Policies.

8.3.1.1 Security-Framework Manager

Der Security Framework Manager liefert dem SCM oder den entsprechenden anderen Security Modulen die Policy-Entscheidungen für den Security-Service. Er wird primär vom SCM aufgerufen, wenn die Applikation eine Kommunikation aufbauen will. Auch andere Module können den manager zur Regulierung ihrer Aktivität aufrufen. Er erstellt die Policy-Entscheidungen für:

- Aufbau eines sicheren Kanal (e.g. IPSec oder TLS);
- Key-Management (e.g. Zeitpunkte für den Rekeying-Prozess, Erstellung neuer Schlüssel…);
- Intrusion Management (Bestimmung der Reaktion im Fall einer Intrusion);
- Authentication Management (welcher Authentikationsprozess für welche Dienste und mit welchen Security Credentials?);

8.3.1.2 Security Application Manager

Der Security Application Manager stellt Policy-Entscheidungen direkt den Applikationen zur Verfügung. Die betroffenen Policies können mehrere Ziele haben:

- Kommunikation zwischen Applikationen;
- Zugriff auf eine Resource (e.g. Schreiben oder Lesen in einer Datenbank);
- Interaktion mit einem Prozess (z.B. Kapazität beschränken oder Prozess stoppen/entblocken).

Für diese Policy-Entscheidung überprüft der Manager in den meisten Fällen in seiner Applikations-ACL, ob die angefragte Aktion erlaubt ist. Falls die Policy komplexer ist, kann es zusätzlich nötig sein, dass den Manager einen Kontext im Kontext Engine zu überprüfen (Analyse der Situation, Bedingung über externe Parameter zu bestätigen, z.B. über die Location, die Zeit…).

8.3.1.3 Policy Database

Im „Policy Definition“ Teil wurde die Vielfalt von Policies dargelegt. Die Speicherung dieser Policies ist ein Kernpunkt des Policy Managements. Grundsätzlich existieren viele verschiedene Formate für Policies. In unserer Architektur definieren wir eine Policy-Datenbank pro Policy-Ziel (e.g. Applikation, KMM, SCM, IPSec…).

8.3.2 Struktur des PEPs

Sicherheitsmechanismen werden auf verschiedenen Schichten des ISO/OSI-Schichtenmodells über Protokoll-spezifische Layer-Enforcement-Points angebunden. Dabei nutzen die Layer-Enforcement-Points einheitliche Schnittstellen der Enforcement-Points des Security-Frameworks. Die implementierungs-spezifischen Layer-Enforcement-Points teilen dem Enforcement-Point des Frameworks die erforderlichen Security-Anforderungen mit. Diese Anforderungen werden dann vom Enforcement Point innerhalb des Frameworks umgesetzt. Innerhalb dieser Enforcement-Points werden, je nach Security-Anforderungen verschiedene interne Security-Mechanismen aufgerufen. Diese Mechanismen werden durch eine Abstraktionsschicht, die Security-Policy-Module, innerhalb des Frameworks eingebunden.

8.4 ANGRIFFSSZENARIEN

Das Autorisierungsmodul ist wie jedes andere Modul anfällig für Angriffe.

8.4.1 Angriffsziele

Angreifer haben mehrere Gründe und Motivationen, dieses Modul zu kompromittieren:

- Erstellen unerlaubter Policies, bzw. modifizieren der Policy-Datenbank;
- Kompromittieren eines Dienst, um die zugeordneten Policies zweckentfremdet zu Angriffen zu benützen (z.B. Gewinn von Zugriffrechten);
- Denial-of-Service: Blockieren des Autorisierungsmoduls für den Zugriff anderer Dienste.

8.4.2 Angriffsmethode

Es gibt für den Angreifer mehrere Wege, diese Ziele zu erreichen:

- Impersonation: Der Angriff fälscht die Authentifizierungsprotokolle und verwendet auf diesem Weg aus Sichtweise des Systems eine autorisierte Identität (Diebstahl oder Erstellung von Identitäten), um Policies zu erstellen oder zu missbrauchen.
- Dienste Kompromittieren auf Software- und Hardwareebene, um die Kontrolle überzunehmen:
  - Angriff auf das Autorisierungsmodul um das Policy-Management kontrollieren;
  - Angriff auf einen anderen Dienst um illegalen Zugriff zu Ressourcen zu bekommen;
- Denial-of-Service Angriff: Der Angriff blockiert den Zugriff zum Autorisierungsmodul (Flooding-Attack, Reset-Attack).

8.4.3 Angriffsauswirkungen auf das System

Kompromittierter Teil des Systems	Auswirkung auf das System
Autorisierungsmodul: Policy-Management Kompromittierung (HW/SW Manipulation)	Kritisch, das gesamte System kann missbraucht werden.

Kompromittierter Teil des Systems	Auswirkung auf das System
Autorisierungsmodul: DoS-Angriff	Kritisch, keine Sicherheitsentscheidung können getroffen werden.
Anderer Dienst: HW/SW Manipulation	Variabel, die Auswirkung hängt vom kompromittierten Dienst ab (Infotainment/ Bremsensystem)

Tabelle 13: PMM- Angriffsauswirkungen

8.4.4 Autorisierungsmodularchitektur und Sicherheit

Zwei Architekturen sind möglich:

- Zentrales System: Alle Policies und das ganze Policy-Management sind auf einer einzelnen Entität im Fahrzeug zusammengefasst. Diese Entität ist ein Single-Point-of-Failure, ein Angriff auf diese Komponente kann den ganzen Autorisierungsprozess verhindern.
- Verteiltes System: Mehrere Entitäten können den Autorisierungsprozess durchführen, es gibt mehrere Policy-Datenbanken, jede für unterschiedliche Domänen spezialisiert. Die Policies sind redundant, so dass nach einem Crash kann ein Modul sein Set von Policies mit Hilfe anderer Module wiederherstellen kann. Ein solches System ist teuerer (redundant) und das Policy-Management ist komplexer, aber resistenter gegen DoS-Angriffe und Fehler (Back-up Möglichkeit).

8.5 FEHLERFÄLLE FÜR DAS AUTORISIERUNGSMODUL

8.5.1 Autorisierungsfehler

8.5.1.1 Autorisierungsfehler und Authentifizierung

Beschreibung	Policy-Anforderer (bzw. Autorisierungsmodul) kann das Autorisierungsmodul (bzw. Policy-Anforderer) nicht authentifizieren.
Gründe	• Fehler im Authentifizierungsprozess • Ungültiges kryptographisches Material (falsches Update, CRL)
Verhalten des Systems	Policy-Anforderer bekommt eine Fehlermeldung und das Autorisierungsmodul verwirft die Anfrage.
Impact auf des System	Variabel, hängt von der beantragten Aktion ab. (Infotainment Dienst/ Bremsen Dienst)
Andere betroffene Sicherheitsmodule	Authentifizierungsmodul

Tabelle 14: Autorisierungsfehler und Authentifizierung

8.5.1.2 Autorisierungsfehler und Policy- Datenbankzugriff

Beschreibung	Das Autorisierungsmodul kann neue Security-Policies nicht lesen, updaten und erstellen.
Gründe	• Fehler in der Policy-Datenbank (Fehler der kryptographischen Funktionen, Fehler im Policy-Speicherprozess, Fehler des Krypto-Moduls (CSM), um die Policy-Schlüssel abzurufen) • Problem in der Authentifizierungs des Hardware-Moduls
Verhalten des Systems	System arbeitet mit der bisherigen Version der Policies, und den Policies-Entscheidungen, die in lokal im PEP gespeichert sind. Sonst erkennt das System das Problem und leitet einen Recovery-Prozess ein.
Impact auf dem System	• Am Bandende: begrenzt, das Problem kann schnell behoben werden. • Während Laufzeit: kritisch. Da einige Autorisierungstickets zeitlich begrenzt sind, müssen sie regelmäßig erneuert werden. Außerdem können Sicherheitsentscheidungen in der Domäne der defekte Datenbank nicht getroffen werden.
Andere betroffene Sicherheitsmodule	Authentifizierungsmodul, CSM

Tabelle 15: Autorisierungsfehler und Policy-Datenbankzugriff

8.5.1.3 Autorisierungsfehler und Erstellung / Interpretation von Policy-Anfragen

Beschreibung	Der MW-Dienst schafft es nicht, eine Policy-Nachfrage zu serialisieren und/oder das Autorisierungsmodul kann sie nicht interpretieren.
Gründe	MW-Problem (Fehler, schlechte Serialisierung, Crash, Versionenkonflikt)
Verhalten des Systems	Der Dienst verschickt die Anfrage nicht oder das Autorisierungsmodul verwirft sie.
Impact auf dem System	• Fehler des Diensts: Variabel, hängt von der beantragten Aktion ab. (Infotainment Dienst/ Bremsen Dienst) • Fehler im Autorisierungsmodul: Kritisch, keine Anfrage bekommt eine Antwort.
Andere betroffene Sicherheitsmodule	MW-Implementierung

Tabelle 16: Autorisierungsfehler und Policy-Generierung und Verständnis

8.5.2 „Failsafe“ Modus und Recovery

8.5.2.1 Fehlerszenario

1) Nach einem Ausfall oder einem Angriff wird ein Fehler im Autorisierungsmodul entweder von dem Modul selbst oder von einem anderen Dienst detektiert.
2) Der Fehler wird analysiert, um festzustellen:
   a) Ob das System ohne diese Funktionalität weiterarbeiten kann (Fault-Tolerant System);
   b) Falls das System nicht weiterarbeiten kann
      i) Der Fehler behoben werden kann (Reboot und Re-Initialisierung, Recovery-Prozess…)
      ii) Falls der Fehler nicht behoben werden kann, wechselt das System in den „Failsafe“ Modus

8.5.2.2 Failsafe Modus

„Failsafe“ Modus erlaubt dem System, den Effekt des Fehlers zu minimieren, das heißt die Ausfälle und Beschädigungen des Gesamtsystems zu begrenzen.

Dafür hat das System einige Alternativen:

- Sicherheitseigenschaften abschalten (keine Autorisierungsprozesse für einige Dienste);
- Kommunikation mit der Außenwelt unterbinden / stoppen;
- Die Laufzeitumgebung verändern;
- Das gesamte System automatisch auf sichere Art und Weise abschalten;

Dieses Modus ist temporär und reversibel, jedoch kann nur der OEM diesen Modus entblocken (siehe Recovery-Prozess).

8.5.2.3 Recovery-Prozess

Dieser Prozess spielt eine Rolle nach einem Dienstfehler oder nach einer Zeit im Failsafe Modus in einer Wartungsstelle. Der Prozess erlaubt dem System, die eine frühere Konfiguration wiederherzustellen.

Der Prozess kann:

- Das defekte Modul analysieren, löschen und fehlerhafte Teile korrigieren.
- Das Modul (ECU) rebooten, um eine neue Initialisierung zu beginnen;
- Die ECU mit einer neuen Firmware flashen (in einer Wartungsstelle).

Zwei Architekturen sind dafür verfügbar:

- Ein zentrales System. Hier ist die Frage eines „Single-Point-of-Failure“ problematisch, und für einen Recovery-Prozess benötigt es eine Back-up Möglichkeit (innere oder äußere Lösung, um die gewünschte Konfiguration zu specihern und später wiederherzustellen);
- Ein verteiltes System. Die Policies sind in unterschiedliche Domänen unterteilt, die Information sind redundant, und nach einem Teilausfall kann der betroffene Dienst seine ganze Konfiguration mit Hilfe der anderen (nicht betroffenen) Dienste rekonstruieren. Allerdings sind Informationenaustausch, Installation und Updates (für Policies) komplexer.

8.6 Anwendungsspezifisches Security API für Autorisierung

Die Security API erlaubt dem Entwickler, die Autorisierungskomponenten zu verwenden (siehe Abbildung 20):

- Um eine Policy-Entscheidung anzufragen;
- Um eine Policy-Entscheidung durchzuführen;
- Um eine Policy zu beantragen (zwischen PDPs);
- Um eine Policy zu erstellen, zu löschen oder zu aktualisieren.

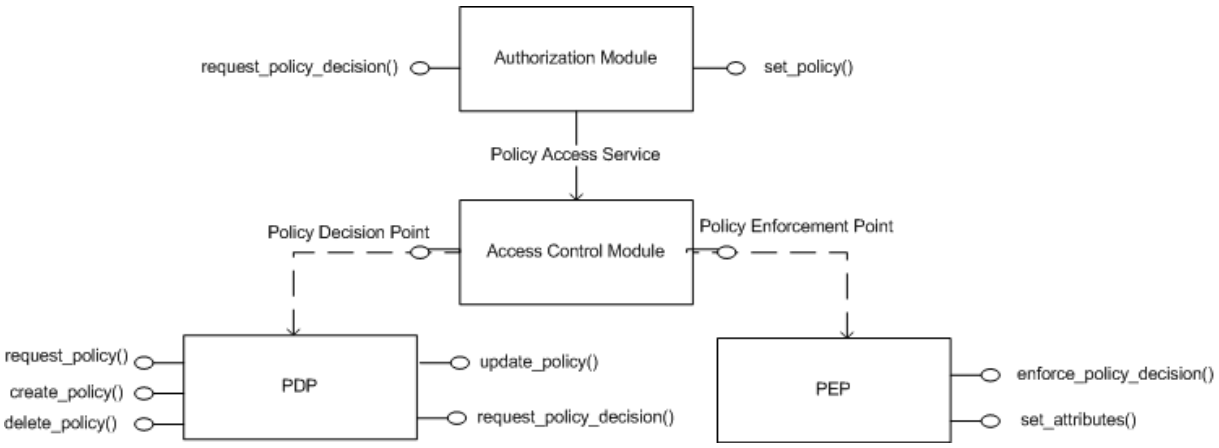

Abbildung 20: Authorization Module Architektur

8.6.1 Funktion: PMM_request_policy_decision() (läuft auf AM)

8.6.1.1 Funktionsbeschreibung

Diese Funktion beantragt eine Sicherheitsentscheidung basierend auf den Policies des gesamten Systems. Diese Funktion wird für jede Sicherheitsentscheidung aufgerufen. Die Anwendung weiß nicht, welchen PDP-Service sie kontaktieren darf und muss. Die AM ist zuständig dafür, die verfügbaren PDP-Location abzubilden und den geeignetsten PDP-Service zu finden.

8.6.1.2 Sicherheitsanforderungen

- Beidseitige Authentifizierung zwischen Authentifizierungsmodul und anforderndem Prozess;
- Kommunikation über einen sicheren Kanal;
- Autorisierung für die Kommunikation (basiert auf einer ACL) ;

8.6.1.3 Input/Output der Funktion

Parameter	Type	Direction	Description
operation	SecurityOperation	In	Requested Operation (String=targetedService.Action.Context)
Security_data	Security_Credentials	In	Security Attributes adapted to the communication security requirements
	ReturnCode	return	„Permission_Granted“, the application receives a positive answer to its decision request, “Granted”. „Permission_Refused“, the application receives a negative answer to its decision request, “Refused” „Permission_error“, An error occured during the processing on the PDP service side „Unkown_Security_Credentials“, the application didn’t provide the right security credentials, the access to the PDP service is refused.

8.6.2 Funktion: PMM_set_policy() (läuft auf AM)

Funktionsbeschreibung

Diese Funktion beantragt die Autorisierung, um eine Policy zu bearbeiten (erstellen, löschen, aktualisieren) und die Veränderungen in die Policy Datenbank zu schreiben. Diese Funktion wird benützt:

- Bei der Fertigung / Erstinitialisierung,um alle Policies zu installieren;
- Nach einem Update, um die veränderten Policies zu schreiben;
- Vom Proxy, um neue dynamische Security-Policies zu installieren.

Die Anwendung weiß nicht, welchen PDP-Service sie kontaktieren darf und muss. Das AM ist zuständig dafür, die verfügbaren PDP-Locations abzubilden und den geeignetsten PDP-Service zu finden.

Sicherheitsanforderungen

- Beidseitige Authentifizierung zwischen Authentifizierungsmodul und anforderendem Prozess;
- Kommunikation über einen sicheren Kanal;
- Autorisierung für die Kommunikation (basiert auf einer ACL) ;

Input/Output der Funktion

Parameter	Type	Direction	Description
Policy_Operation	String	In	Requested Operation to proceed in the policy database (create, delete, update …)
operation	SecurityOperation	In	Policy to set (ID or description)
Security_data	Security_Credentials	In	Security Attributes adapted to the communication security requirements
	ReturnCode	return	„Operation_Completed“, the new policy was successfully integrated in the policy database. „Error_UnreadableRequest“, the policy integration is refused, the new policy isn’t readable. „Error_Processing“,the policy integration is refused, an error occurred during the processing of the function. “Unauthorized_source”, the policy integration is refused, because the source of the policy (requester) is not a valid/authorized policy editor. „Operation_Refused“, the policy integration is refused (example the security credential aren’t valid, no respect of the security communication protocol).

8.6.3 Funktion: PMM_request_policy_decision() (läuft auf PDP)

Funktionsbeschreibung

Diese Funktion beantragt eine Policy-Entscheidung basierend auf lokal installierten Policies oder von anderen PDP. Diese Funktion wird vor jeder Sicherheitsentscheidung verwendet, der PEP erfragt vom PDP eine Sicherheitsentscheidung.

Sicherheitsanforderungen

- Beidseitige Authentifizierung zwischen PDP und anforderendem Prozess;
- Kommunikation über einen sicheren Kanal;
- Autorisierung für die Kommunikation (basiert auf einer ACL) ;

Input/Output der Funktion

Parameter	Type	Direction	Description
pdp_service	Entity_ID	In	Name of the requested and adapted PDP
operation	SecurityOperation	In	Requested Operation
Security_data	Security_Credentials	In	Security Attributes adapted to the communication security requirements
	ReturnCode	return	„Permission_Granted“, the AM receives a positive answer to its decision request, “Granted”. „Permission_Refused“, the AM receives a negative answer to its decision request, “Refused” „Permission_error“, An error occured during the processing on the PDP service side „Unkown_Security_Credentials“, the application didn’t provide the right security credentials to the AM, the access to the PDP service is refused. „Operation_Refused“, the policy integration is refused (example the security credential aren’t valid, no respect of the security communication protocol).

8.6.4 Funktion: PMM_delete_policy() (läuft auf PDP)

Funktionsbeschreibung

Diese Funktion löscht Policies in der Policy Datenbank. Sie wird benützt:

- Bei der Fertigung / Erstinitialisierung, um die Funktionalität zu testen;
- Nach einem Update, um veraltete Policies zu löschen;
- Vom Proxy, um Policies für nicht mehr am Auto angeschlossene externe Geräte zu löschen.

Sicherheitsanforderungen

- Beidseitige Authentifizierung zwischen PDP und anforderendem Prozess;
- Kommunikation über einen sicheren Kanal;
- Autorisierung für die Kommunikation (basiert auf einer ACL) ;

Input/Output der Funktion

Parameter	Type	Direction	Description
pdp_service	Entity_ID	In	Name of the requested PDP
operation	SecurityOperation	In	Policy to delete (ID or description)
Security_data	Security_Credentials	In	Security Attributes adapted to the communication security requirements

Parameter	Type	Direction	Description
	ReturnCode	return	„PolicyDeletion_Completed“, the new policy was successfully deleted in the policy database. „Error_UnreadableRequest“, the policy deletion is refused, the new policy isn’t readable. „Error_Processing“, the policy deletion is refused, an error occurred during the processing of the function. “Unauthorized_source”, the policy deletion is refused, because the source of the policy (requester) is not a valid/authorized policy editor. „PolicyDeletion_Refused“, the policy integration is refused (example the security credential aren’t valid, no respect of the security communication protocol).

Notiz: die Funktionen create_policy(), update_policy() funktionieren in dem gleichen Weg.

8.6.5 Funktion: PMM_request_policy() (läuft auf PDP)

Funktionsbeschreibung

Diese Funktion beantragt eine Policy bei einem anderen PDP und schreibt sie in seine Datenbank. Diese Funktion wird verwendet, falls der erste kontaktierte PDP nicht fähig ist, zu antworten, und er einen anderen bekannten PDP verwendet, der eine Entscheidung treffen kann.

Sicherheitsanforderungen

- Beidseitige Authentifizierung zwischen PDP und anforderendem Prozess;
- Kommunikation über einen sicheren Kanal;
- Autorisierung für die Kommunikation (basiert auf einer ACL) ;

Input/Output der Funktion

Parameter	Type	Direction	Description
pdp_service	Entity_ID	In	Name of the requested PDP
operation	SecurityOperation	In	Description of the Requested Operation without the context, the target is optional
Security_data	Security_Credentials	In	Security Attributes adapted to the communication security requirements
operation	Securityoperation	Out	Policy with context and target explicit
	ReturnCode	return	“policy_Granted”, the policy(ies) is (are) sent to the PDP requester. „Error_UnreadableRequest“, the policy retrieval is refused, the policy request isn’t readable. „Error_Processing“, the policy retrieval is refused, an error occurred during the processing of the function. „Policy_Refused“, the policy integration is refused (example the security credential aren’t valid, the requester isn’t allowed, no respect of the security communication protocol…).

8.6.6 Funktion: PMM_enforce_policy_decision() (läuft auf PEP)

Funktionsbeschreibung

Diese Funktion führt die Entscheidung des PDP aus. Die Funktion analysiert die Entscheidung des PDPs, vermittelt sie zur Anwendung und setzt die Entscheidung durch. Die Funktion betrifft auch Anwendungen, die selbst keine Funktion für eine Policy-Decision Anfrage aufgerufen hat, aber deren Prozess einen Einfluss auf die Policy-Decision hat. Eine Policy kann erfordern, mit solchen Prozessen zu interagieren. Die Durchsetzung dieser „Enforce“-Funktion kann einen Prozess beenden oder die Kapazitäten einer Anwendung beschränken.

Sicherheitsanforderungen

- Beidseitige Authentifizierung zwischen PEP und anforderendem Prozess;
- Kommunikation über einen sicheren Kanal;
- Autorisierung für die Kommunikation (basiert auf einer ACL) ;

Input/Output der Funktion

Parameter	Type	Direction	Description
Target_Entity	Entity_ID	In	Entity, on which the policy decision has to be enforced
decision	OperationDecision	In	Operation (policy decision), that has to be enforced
Security_data	Security_Credentials	In	Security Attributes adapted to the communication security requirements
	ReturnCode	return	„Enforcement_Succeed“, the policy decision has been enforced on the target entity. „Enforcement_Failed“, the enforcement failed, the target entity refused the policy decision, or the PEP didn’t manage to force its decision (e.g. killing, blocking the process) „Enforcement_error“, the enforcement failed, an error occurred during the processing of the function.

8.7 SECURITY PLUG-IN FÜR AUTORISIERUNG

8.7.1 Policy Manager

Das Modell der verteilten PDP bietet Flexibilität, erhöht aber die Komplexität der Suche nach der geeignetsten Policy. Dieses Plug-in ist zuständig dafür, die physische Verwaltung der Policy-Datenbank (TLS-Policy, IPSec-Policy, Application-Policy, KMM-Policy…) zu bewältigen. Das Plug-in liefert:

- Schnittstelle für die direkte Policy-Einfügung. Der OEM kann direkt die Policies aktualisieren. Die Policies werden vom Plug-in in der geeigneten Datenbank gespeichert;
- Management von Sicherheitseigenschaften: Das Plug-in ist fähig, die Sicherheitsinformationen der Policies (e.g. digitale Zertifikate während eines Updates) zu überprüfen und den Ablauf ihrer Gültigkeit oder Annullierung zu verwalten;
- Effiziente Service-Discovery-Mechanismen, um den PDP-Service zu unterstützen. Das Plug-in erstellt eine Mapping von Policies und dann kann für jede PDP-Nachfrage die optimale Policy zurückliefern.
- Bei externer Kommunikation ist das Plug-in verantwortlich für die Generierung von neuen Policies (e.g. für Filter). Diese neuen Policies basieren auf Informationen vom AMM (über die Authentifizierung von externen Geräten) und den Anforderungen der internen Anwendungen auf die zugegriffen wird. Dieser Teil des Plug-ins wird auf dem Proxy verwaltet bzw. ausgeführt.

8.7.2 Policy Decision Manager

Dieses Plug-in ist zuständig dafür, die Sicherheitsentscheidung direkt in einem geeigneten Policy-Format für das Sichere Kanal-Modul oder die Anwendung zu generieren. Das Plug-in kann als ein wesentlicher Bestandteil des PDPs betrachtet werden. Als Eingabe nimmt es eine vom Policy-Manager gespeicherte Policy, sowie andere Informationen aus dem AMM und aus anderen kontextsensitiven Infrastrukturen des Fahrzeugs und erzeugt die Policy-Entscheidung. Dieses Plug-in ist fähig, mehrere Policy- Formate zu verstehen und mögliche Policy-Konflikte (e.g. Prioritätsregeln für Policy) aufzulösen.

8.7.3 Policy Converter

Dokumente in Policy-Sprache (z.B. XACML, EPAL) können umfangreich, komplex, und schwierig zu verstehen sein. Die Interpretation kann längere Verarbeitungszeit und vielleicht eine Busüberlastung zur Folge haben. Die Speicherung auf einer ECU kann auch problematisch werden. Das Autorisierungsmodul braucht daher ein leistungsfähiges Übersetzungssystem.

Der PDM muss einen Plug-In-Mechanismus bieten, der die Policy-Beschreibungen (z.B. XACML) auf PDM-Sprache übersetzt. Der OEM kann also alle Policies in einer Policy-Sprache vorkonfigurieren. Dafür wird die Funktion set_Policy() der Security-API benützt, als Input nimmt sie die Policy, die sie übersetzt und auf die betroffene PDPs installiert (Push-Methode). Alternativ können diese Policies durch die Funktion get_Policy() (Pull-Methode) von den PDPs angefragt werden (Abbildung 21). Dieses Plug-In funktioniert für Autorisierungstickets und Konfigurationstickets. Das Plug-In muss wie alle andere Komponenten des Systems über einen sicheren Kanal kommunizieren.

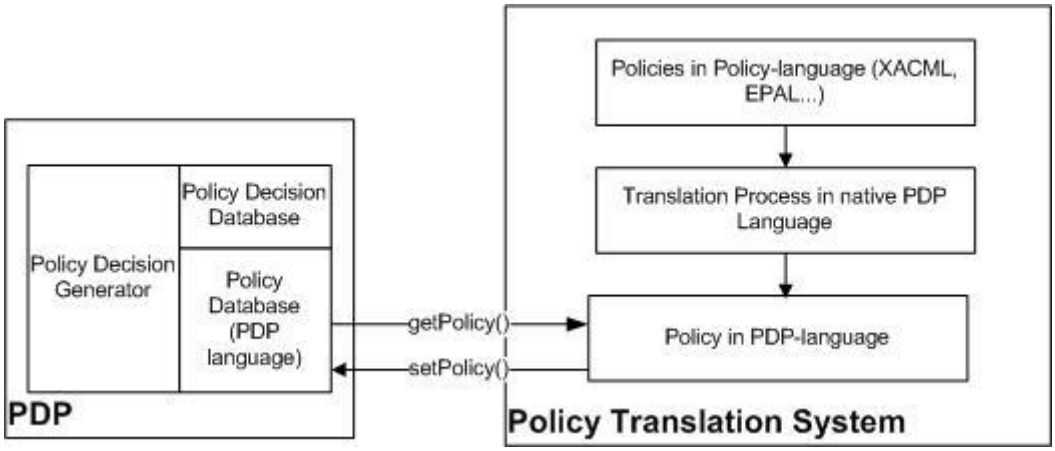

Abbildung 21: Plug-In: Übersetzungsprozess

9. INTRUSION-MANAGEMENT-MODUL

Intrusion Detection Systeme wurden innerhalb von SEIS bereits in AP4.2 betrachtet. Im Folgenden werden die Schnittstellen der Intrusion Detection zum Rest der Middleware betrachtet. Die Schnittstellen der fahrzeugexternen Kommunikation werden im Rahmen von SEIS in AP4.4 behandelt.

Innerhalb der Middleware kümmert sich das Intrusion Management Modul (IMM) um die Intrusion Detection. Da dieses Modul zur Angriffserkennung weitestgehend passiv ist, ist es nicht nötig, aus der Applikation heraus auf dieses Modul zuzugreifen. Hinter dem IMM können in der Praxis verschiedene Arten von Intrusion Detection Systemen implementiert sein. Idealerweise sind diese verschiedenen Systeme aufeinander abgestimmt und ergänzen sich gegenseitig.

9.1 ANWENDUNGSSPEZIFISCHES SECURITY API

Neben der automatisierten Erkennung von Angriffen auf der Ebene des Flashspeichers oder der Netzwerkkommunikation können andere Module des Security Frameworks das IMM auf die Möglichkeit eines Angriffs hinweisen. Dazu wird vom IMM die Funktion „notifyViolation()“ bereitgestellt. Ein Beispiel ist: Das SCM soll eine Verbindung öffnen, die jedoch vom PDM verboten ist. Das SCM kann diesen Vorfall beim IMM melden.

Das IMM wertet je nach Einstellung die verfügbaren Daten aus und meldet die Ergebnisse an eine zentrale ECU, deren IMM als zentrale Auswertungskomponente dient. Hierzu wird die Funktion „receiveAnalysisData()“ der zentralen Komponente aufgerufen.

Weiter besitzt das IMM die Möglichkeit, Änderungen an den internen Einstellungen durchzuführen. Die dafür benötigte Funktionalität wird von der Funktion „changeConfiguration()“ des IMM bereitgestellt.

Ein Update der Signaturdatenbank des IMM kann mit Hilfe der Funktion „updateSignatures()“ angestoßen werden.

9.1.1 Funktion: notifyViolation()

- Funktionsbeschreibung: Diese Methode erlaubt das Melden von sicherheitsrelevanten Ereignissen von verschiedenen Modulen des Security Frameworks. Sie kann von jedem Modul oder jeder Applikation aufgerufen werden, welche eigene Daten zur Analyse bereitstellen kann. Entsprechend nötige Überprüfungen der Berechtigung werden beim Policy Management durchgeführt. Darüberhinaus muss das IMM die Eingaben der Module erst auf Plausibilität prüfen, da diese Funktion durchaus auch von einem Angreifer missbraucht werden kann. Natürlich muss jede Meldung authentifiziert und integer durchgeführt werden.
- Input/Output-Format:

Parameter	Type	Direction	Description
module_id	MODULE_ID	In	Identität des sendenden Moduls
module_type	MODULE_TYPE	In	Art des sendenden IDS
event_type	EVENT_TYPE	In	Art des gemeldeten Events
event_data	BINARY	In	Zusätzliche Informationen über das gemeldete Event • Initator einer Verbindung • Ziel der Verbindung • Uvm.
	ReturnCode	return	„IDS_NOTIFY_SUCCESSFUL“: Übertragung erfolgreich „IDS_NOTIFY _FAILED“: Übertragung nicht erfolgreich

9.1.2 Funktion: receiveAnalysisData()

- Funktionsbeschreibung: Diese Methode erlaubt das Senden der Analyseergebnisse an ein hierarchisch höherliegendes zentrales System. Sie kann von jedem Modul oder jeder Applikation aufgerufen werden, welche eigene Daten zur Analyse bereitstellen können. Entsprechend nötige Überprüfungen der Berechtigung werden beim Policy Management durchgeführt. Die Kommunikation muss authentifiziert und integer durchgeführt werden.
- Input/Output-Format:

Parameter	Type	Direction	Description
ids_id	IDS_ID	In	Identität des sendenden IDS
ids_type	IDS_TYPE	In	Art des sendenden IDS
ids_data	BINARY	In	Analysedaten des sendenden IDS
	ReturnCode	return	„IDS_PARSE_DATA_SUCCESSFUL“: Übertragung erfolgreich „IDS_PARSE_DATA_FAILED“: Übertragung nicht erfolgreich

9.1.3 Funktion: changeConfiguration()

- Funktionsbeschreibung: Diese Methode erlaubt das Ändern der Konfiguration des IMM. Sie kann von jedem Modul oder jeder Applikation aufgerufen werden, welches zur Änderung der Daten autorisiert ist und sich gegenüber dem System authentifizieren kann. Entsprechend nötige Überprüfungen der Berechtigung werden beim Policy Management durchgeführt.
- Input/Output-Format:

Parameter	Type	Direction	Description
ids_id	IDS_ID	In	Identität des sendenden IDS
ids_type	IDS_TYPE	In	Art des empfangenden IDS
ids_data	BINARY	In	Neue Konfigurationsdaten für das IDS
	ReturnCode	return	„IDS_PARSE_CONFIG_SUCCESSFUL“: Übertragung erfolgreich „IDS_PARSE_CONFIG_FAILED“: Übertragung nicht erfolgreich

9.1.4 Funktion: updateSignatures()

- Funktionsbeschreibung: Diese Methode erlaubt es ein manuelles Update der Signaturdatenbank des IDS anzustoßen. Die Signaturen werden daraufhin vom Zielsystem selbständig angefordert. Sie können dabei entweder fahrzeugintern vorgehalten werden oder von einem Server des OEM heruntergeladen werden. Entsprechend nötige Überprüfungen der Berechtigung werden beim Policy Management durchgeführt.
- Input/Output-Format:

Parameter	Type	Direction	Description
ids_id	IDS_ID	In	Identität des sendenden IDS
ids_type	IDS_TYPE	In	Art des empfangenden IDS
	ReturnCode	return	„IDS_PARSE_SIG_SUCCESSFUL“: Übertragung erfolgreich „IDS_PARSE_SIG_FAILED“: Übertragung nicht erfolgreich

9.2 SCHNITTSTELLEN ZU ANDEREN MODULEN DER MIDDLEWARE

Um nach einem erkannten Angriff die Funktionalität des Systems anzupassen verwendet das IMM die Funktionalitäten der Module: PDM, KMM und SCM. Da eine Konfigurationsänderung innerhalb der Middleware die Unbenutzbarkeit des Fahrzeugs zur Folge haben kann, sind diese Mittel nur mit Vorsicht einzusetzen.

9.2.1 Schnittstelle des IMM zum PDM

Um nach einem erkannten Angriff die Policies eines Systems anzupassen verwendet das IMM die Funktion „set_Policy()“ des PDM.

9.2.2 Schnittstelle des IMM zum KMM

Um nach einem erkannten Angriff die Schlüssel eines Systems zu Sperren verwendet das IMM die Funktion „revoke_Key()“ des KMM.

9.2.3 Schnittstelle des IMM zum SCM

Um während einem aktiven Angriff Verbindungen eines Systems trennen zu können, verwendet das IMM die Funktion „close_Connection()“ des SCM.

10. IT-SICHERHEITS-FUNKTIONEN FÜR DIENSTE DER SEIS-MIDDLEWARE

Die folgende Sektion beschreibt die Sicherheitsspezifikation im Bezug auf Middleware-Dienste (RPC, Publish/Subscribe und Service discovery) für interne und externe Kommunikation. Für jeden der Dienste werden Angreifer und mögliche Angriffe definiert und ihre Auswirkungen auf das System aufgelistet. Anschließend werden basierend darauf die enstprechenden Sicherheitsanforderungen definiert und erste Spezifikationen zu ihrer Realisierung vorgeschlagen.

10.1 RPC

Remote Procedure Calls (RPC) werden genutzt, um Funktionen auf Geräten auszuführen, die nicht zwingend identisch sind mit dem Gerät, dass die Funktion aufruft. Es kann Funktionen geben, die Rückgabewerte erzeugen oder Funktionen die nur eine Aktion auslösen.

Um einen RPC auszuführen muss es eine Möglichkeit geben den Funktionsaufruf, mögliche Parameter der Funktion und mögliche Rückgabewerte über ein Netzwerk zu übertragen. Für das Konzept RPC gibt es keine Bindung an ein Transportprotokoll, wie TCP oder UDP.

Ein RPC wird immer durch einen so genannten *Caller* initiiert und durch den so genannten *Callee* ausgeführt.

Der RPC-Mechanismus kann im Rahmen der in Kapitel 2 definierten Anwendungsfälle eine Einsatzmöglichkeit finden. Hierbei sind die folgenden Anwendungsfälle relevant:

- **Software-Download**: Die Steuerung eines Software Downloads ist über RPC realisierbar. Die eigentlichen Daten sollten jedoch out-of-band (ein gesonderter Kanal/kein RPC) übertragen werden, so dass kein Protokoll für die Übertragung großer Datenmengen über RPC erarbeitet werden muss.
- **CE-Geräte-Integration**: Über das CE-Gerät können Funktionen des Fahrzeuginfotainment gesteuert werden. Weiterhin ist es denkbar, dass mittels des Fahrzeuginfotainments Funktionen auf dem CE-Gerät gesteuert werden.
- (**Remote) Diagnose**: Im Rahmen der Remote Diagnose müssen Konfigurationen vorgenommen werden, welche mittels eines RPC-Mechanismus durchgeführt werden können.
- **Rückfahrkamera**: Die Ansteuerung der Rückfahrkamera kann über einen RPC-Mechanismus erfolgen.

Allgemein können alle Funktionen, die auf anderen Steuergeräten ausgeführt werden, durch RPC realisiert werden.

Die in AP4.1 näher betrachteten Technologien CORBA, Apache Etch, Ice und SLAP bieten alle einen RPC Mechanismus. Die folgenden Beschreibungen sollten also auf alle vier Technologien übertragbar sein.

10.1.1 Voraussetzungen

Um einen RPC nutzen zu können, muss entweder eine Verbindung zwischen dem Caller und dem Callee existieren oder das Ziel für einen RPC muss hinreichend bekannt (adressierbar) sein.

Wenn bereits eine Verbindung zwischen Caller und Callee besteht, kann diese verwendet werden, um den RPC und mögliche Antworten zu übertragen. Sollte keine Verbindung bestehen, aber eine Verbindung möglich sein, sollte die Verbindung hergestellt werden.

Sollte keine Verbindung möglich sein (Caller und/oder Callee unterstützen nur UDP), so kann ein RPC verbindungslos übertragen werden. Dabei muss der Caller dem Callee bekannt sein, bzw bekannt gemacht werden (Service Discovery). Um die im nächsten Abschnitt erläuterte Ausführungsart „at-most-once“ zu erreichen, müssen dabei weitere Vorkehrungen getroffen werden.

10.1.2 Ausführungsarten

Es gibt unterschiedliche Arten RPC auszuführen, bzw. auf Fehler während der Übertragung zu reagieren. Die Ausführungsarten und die entsprechenden Fehlerbehandlungen sind in Tabelle 17 ausgeführt.

RPC Ausführung	Fehlerbehandlung	Bemerkung
maybe	Keine erneute Übertragung bei Fehler	Eine Funktion kann einmal, mehrmals oder auch gar nicht ausgeführt werden.
at-least once	Bei Fehler erneute Übertragung	Funktionen können mehrfach ausgeführt werden.
at-most once	Bei Fehler erneute Übertragung	Es muss sichergestellt werden, dass eine Wiederholung eines RPC als solche erkannt wird, damit keine Funktion doppelt ausgeführt wird.
exactly once	Bei Fehler erneute Übertragung	Es kann nie sichergestellt werden, dass ein Service verfügbar ist, daher ist exactly once nur mit Einschränkungen umsetzbar.

Tabelle 17: RPC Ausführungsarten

In SEIS sollte die Ausführungsart „at-most once“ angestrebt werden, um eine mehrfache Ausführung von Funktionen und damit die Möglichkeit von Replayangriffen zu vermeiden. Um „at-most once“ zu erreichen ist die Erkennung von Fehlern und die eindeutige Identifizierung eines RPC nötig und muss durch das Protokoll, welches für RPC verwendet wird, ermöglicht werden. Die eindeutige Identifizierung ist nötig, um zu verhindern, dass Funktionen doppelt ausgeführt werden. Eine Fehlererkennung ist nötig, um sicherzustellen, dass der RPC angenommen wurde und ein mögliches Ergebnis beim Aufrufer eingetroffen ist. Die Fehlererkennnung sollte sich auf Timeouts beschränken, da verfälschte oder fehlerhafte RPC bereits durch das Security Framework erkannt werden sollten.

10.1.3 Fehlererkennung

Wird für die Übertragung eines RPC TCP verwendet, können Netzwerktimeouts als Fehlererkennung ausreichen. Da das Netz in SEIS bekannt ist, können diese Timeouts entsprechend klein gehalten werden. Fehler bei der Übertragung werden bereits durch das Protokoll und möglicherweise durch unterliegende Integritätsprüfungen behandelt und müssen nicht explizit durch den RPC Mechanismus erkannt werden.

Wird ein Protokoll verwendet, welches nicht Fehler resistent ist, muss die Integrität der Nachrichten explizit - durch zum Beispiel eine Signatur oder HMAC - sichergestellt werden. Auch diese Prüfungen sollten nicht Teil des RPC Mechanismus sein, sondern von der Middleware zur Verfügung gestellt werden.

10.1.4 Synchrone vs. asynchrone Ausführung

Ein RPC kann für einen Caller sowohl synchron als auch asynchron erfolgen. Während eines synchronen RPC wartet der Caller auf den Abschluss des RPC, bevor das Programm fortgesetzt werden kann. Dabei kann es zu Wartezeiten aufgrund von Fehlern kommen, die das Protokoll selbstständig zu beheben versucht. Bei einem asynchronen RPC kann der Caller in der Zwischenzeit weiterarbeiten und z.B. über einen Callback-Mechanismus über das Ergebnis des RPC informiert werden. Solange keine weiteren Funktionen von dem Ergebnis des RPC abhängen entstehen auch bei Fehlern keine Wartezeiten.

Ob ein RPC in SEIS synchron oder asynchron ausgeführt wird, muss anwendungsfallspezifisch entschieden werden. Es ergäbe zum Beispiel keinen Sinn das komplette Userinterface (inklusive Media Player u.ä.) zu blockieren, während per RPC eine Routenberechnung für die Navigation durchgeführt wird.

Während per RPC eine Mediendatei gestartet wird, kann das System jedoch für eine bestimmte Zeit blockiert sein.

Sollte das Protokoll, welches für SEIS eingesetzt wird, keinen asynchronen RPC unterstützen, so könnte die Asynchronität in der Middleware umgesetzt werden. Es ist in der Middleware möglich für jeden RPC, der asynchron ablaufen soll einen eigenen Programmthread zu starten und damit Asynchronität für eine Anwendung transparent zu simulieren. Sollte das Protokoll gar keine Unterstützung für ein solches Vorgehen bieten, muss auf eine asynchrone Ausführung von RPC verzichtet werden. Sollte das Protokoll nur asynchrone RPC zulassen, kann Synchronität in der Middleware simuliert werden, indem vor der Rückkehr aus der API Funktion die Antwort des Callees abgewartet wird.

10.1.5 Ablauf eines RPC

Der Ablauf eines RPC aus Sicht des Callers ist in Abbildung 22 dargestellt. Bei Verbindungslosen RPC trifft nur der blau hinterlegt Bereich zu.

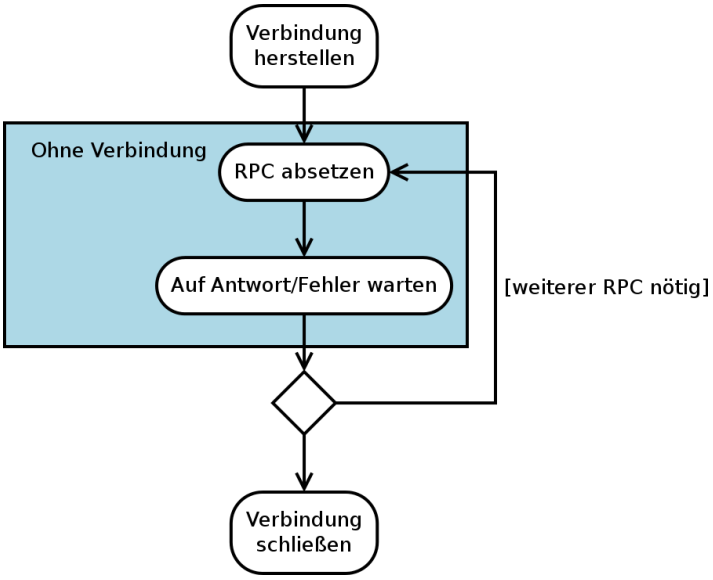

Abbildung 22: RPC Ablauf aus Sicht der aufrufenden Entität

10.1.6 RPC innerhalb des Fahrzeugs

Innerhalb des Fahrzeugs ist die Konfiguration bekannt und statisch. Den vorhandenen Steuergeräten sind die verfügbaren Services und deren RPC Fähigkeiten bekannt. Sie müssen daher nicht dynamisch über ein Service Discovery erkannt werden. Beim Austausch von Steuergeräten gibt es eine Anlernphase, während der die Konfiguration und die verfügbaren Mechanismen bekannt gemacht werden.

10.1.7 RPC zwischen Fahrzeug und Außenwelt

Im folgenden Abschnitt wird ein CE-Gerät als Beispiel verwendet. Jedoch treffen die Aussagen zum Beispiel auch auf (Remote) Diagnose zu.

Um Funktionen zu realisieren, die ein CE-Gerät im Fahrzeug nutzen können soll, müssen RPC zwischen der Außenwelt und dem Fahrzeugnetz geroutet werden. Dieses Routing übernimmt der im Rahmen der SEIS Architektur definierte Security Proxy, der in Kapitel 10.2.2.3 aus Sicht der Middleware beschrieben wird.

Das CE-Gerät bzw. die Anwendung auf dem CE-Gerät muss sich gegenüber dem Security Proxy authentifizieren, um Zugriff auf die Funktionalitäten im Fahrzeug zu erhalten. Eine Authentifizierung zwischen

dem Service im Fahrzeug und dem CE-Gerät findet in der Regel nicht statt, da diese im Normalfall nicht direkt miteinander kommunizieren.

Die Funktionalitäten, die der Security Proxy (und damit das Fahrzeug) anbietet, kann das CE-Gerät nach der Authentifizierung per Service Discovery erkennen. Siehe dazu Kapitel 10.3.

Im Normalfall nimmt das CE-Gerät die Rolle des Callers ein. Das bedeutet, das CE-Gerät möchte Funktionen im Fahrzeug ausführen und das Ergebnis der Funktionen erfahren. Der Ablauf dazu ist in Abbildung 23 dargestellt.

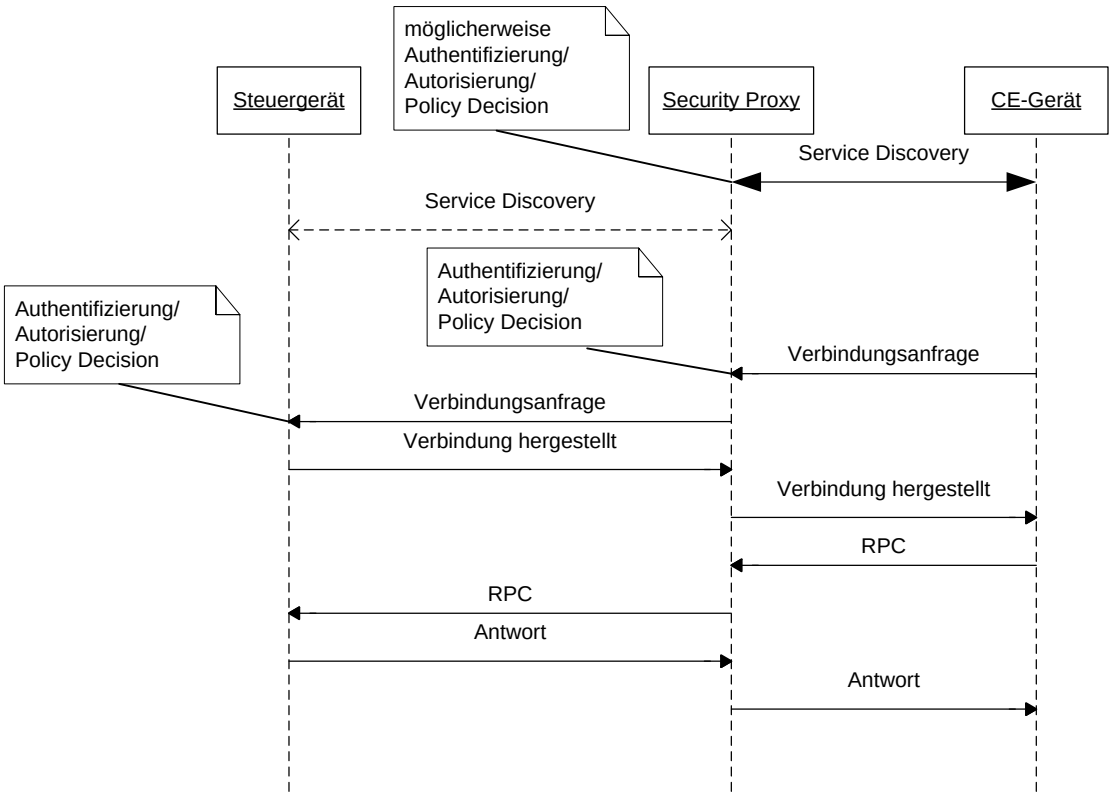

Abbildung 23: RPC: CE-Gerät Caller

Der andere Fall ist, dass das CE-Gerät als Callee fungiert und aus dem Fahrzeug heraus Funktionen auf dem CE-Gerät aufgerufen werden sollen. Für diesen Fall muss das CE-Gerät einen Dienst (mit RPC-Funktionalität) seinerseits bereitstellen. Dieser Dienst kann dann durch den Proxy von Steuergeräten im Fahrzeug in Anspruch genommen werden.

Der Proxy hat im Allgemeinen eine feste Menge an Diensten, die er für ein CE-Gerät im Fahrzeug verfügbar machen kann. Je nach Anwendungsfall kann es nötig sein, dass ein Steuergerät selbstständig bei Verfügbarwerden eines Dienstes die Verbindung zu diesem Dienst herstellt. Dafür muss über ein Publish/Subscribe Mechanismus die Verfügbarkeit des Dienstes im Fahrzeug bekannt gemacht werden.

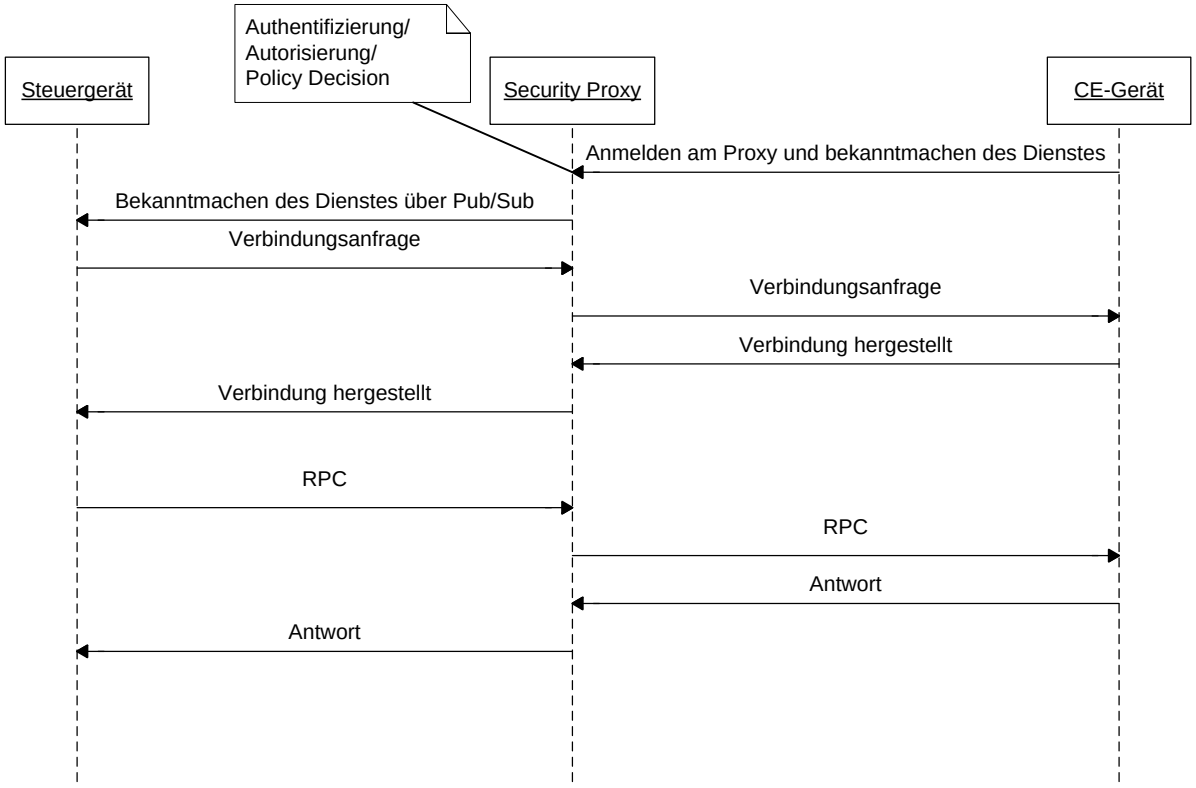

Abbildung 24: RPC: CE-Gerät Callee

10.1.8 RPC Security Anforderungen

Die Security Anforderungen an den RPC erstrecken sich über mehrere Bereiche:

Authentifizierung: Mindestens der Client muss sich gegenüber dem Server authentifizieren und damit die Berechtigung für die Nutzung der Funktion erbringen. Idealerweise sollte sich auch der Server gegenüber dem Client authentifizieren und damit seine Unversehrtheit nachweisen. Dies ist notwendig, damit der Client identifizieren kann, ob er dem Ergebnis der Funktion vertrauen kann.

Autorisierung: Der Server darf nur die Anfragen beantworten, welche tatsächlich von einem authentifizierten und darauf aufbauend autorisierten Client stammen. Hierdurch wird verhindert, dass ein unautorisierter Teilnehmer unberechtigt Funktionen anfordern kann.

Integrität: Die Kommunikation zwischen Client und Server muss integritätsgesichert erfolgen. Die Daten dürfen nicht unbemerkt verändert werden können.

Vertraulichkeit: Die Vertraulichkeit der RPCs muss nicht generell sichergestellt werden. Dies ist Use-Case abhängig zu entscheiden.

Nichtabstreitbarkeit: Nichtabstreitbarkeit muss nicht im eigentlichen Sinn sichergestellt sein. Es gnügt eine Nachverfolgbarkeit vorzusehen, welche z.B. über Logging realisiert werden kann. Eine RPC-Anfrage an einen Client sowie auch die RPC-Antwort eines Servers sollten demnach nachverfolgbar sein. Die Verfolgbarkeit von gestellten Anfragen und entsprechenden Antworten (speziell bei verändernden Funktionen) muss jederzeit gegeben sein.

Verfügbarkeit: Die Verfügbarkeit des RPC-Mechanismus sollte im Rahmen der technischen Möglichkeiten sichergestellt werden. Die Serverantwort auf einen RPC muss bei synchronen Aufrufen sichergestellt sein.

Die Anforderungen werden durch das Security Framework erfüllt, so dass eine Anwendung, die RPC nutzt, transparent auf die von ihr benötigten Dienste zugreifen kann. Vorausgesetzt sie hat die nötigen Berechtigungen.

10.1.9 RPC API

Die API, die durch die Middleware für einen RPC Caller zur Verfügung gestellt wird, sollte aus folgenden Funktionen bestehen:

- **rpc_callSync**(connectionHandle, function-identifier, parameters, result)
  - connectionHandle: (input) Handle oder Identifikation für eine Verbindung, die bereits mit dem Dienst hergestellt wurde.
  - function-identifier: (input) Identifiziert die Funktion, die aufgerufen werden soll.
  - parameters: (input) Eventuelle Parameter, die an die Funktion übergeben werden sollen.
  - result: (output) Eventuelle Rückgabewerte der Funktion.
- **rpc_callAsync**(connectionHandle, function-identifier, parameters, callback)
  - callback: (input) Funktionszeiger auf eine Funktion, die asynchron aufgerufen werden soll, wenn die Funktion abgeschlossen ist. Der callback erhält als Parameter die Rückgabewerte (falls vorhanden) der Funktion.

Verbindungsauf- und abbbau sind unabhängig von der Art des Dienstes zu behandeln und werden durch die Middleware separat zur Verfügung gestellt.

10.2 PUBLISH/SUBSCRIBE DIENST

Publish/Subscribe-Dienste (kurz: Pub/Sub-Dienste) werden primär verwendet, um zentral zur Verfügung stehende, dynamische Daten für potentiell viele Interessenten als Push-Dienst verfügbar zu machen und aktuell zu halten. Beispiele im Fahrzeug sind etwa die aktuelle Fahrzeuggeschwindigkeit, die sowohl für die Anzeige im Kombi-Instrument als auch für diverse Fahrerassistenzsysteme, das Navigationssystem, ggf. das System zum Öffnen des Kabrioverdecks und zukünftig evtl. auch externe CE-Geräte interessant ist. Ein Pub/Sub-Dienst ermöglicht es dem Dienst- bzw. Datenanbieter (“Publisher”), für den angebotenen Dienst eine Liste von Interessenten (“Subscribern”) zu halten und ggf. anzupassen, und umgekehrt den Subscribern, Datendienste zu abonnieren, also bspw. über Änderungen des Zustands informiert zu werden. Die Aktualisierung der Daten kann hierbei nach verschiedenen Strategien organisiert sein, etwa ein regelmäßiges Versenden von Updates unabhängig vom vorliegenden Inhalt (zeitgesteuert) oder ein Versenden bei Änderung des Wertes (ereignisgesteuert).

Speziell interessant ist ein Pub/Sub-Dienst naheliegenderweise, wenn die Gruppe der Interessenten eines Dienstes nicht statisch und zur Designzeit bekannt ist, sondern sich dynamisch verändert und im Betrieb angepasst werden muss. Dies ist im Automobilbereich speziell dann der Fall, wenn zusätzlich zur Werksausstattung später externe Geräte an- und eingebunden werden sollen. Da dieser Fall auch aus Security-Sicht spezielle Anforderungen hat, wird im weiteren Verlauf des Kapitels darauf vertieft eingegangen.

10.2.1 Funktionalitäten und Modellierung

Für die Realisierung und Absicherung eines Pub/Sub-Dienstes werden verschiedene interagierende Funktionalitäten benötigt. Die einzelnen Funktionalitäten und ihr Zusammenspiel werden hier getrennt betrachtet. Je nach Einsatzszenario ist die gemeinsame Implementierung verschiedener Funktionalitäten auf einer physikalischen Entität denkbar und sinnvoll, allerdings kann die Aufteilung je nach Einsatzzweck variieren. Die getrennte Betrachtung erhöht damit die Flexibilität der weiteren Designentscheidungen.

Es werden die folgenden Funktionalitäten betrachtet:

- **Publisher (Publication Server):** Der Publisher stellt den Datendienst bereit und die entsprechenden Informationen angemeldeten Interessenten (Subscribern) zur Verfügung. Dafür ist zu jedem betrachteten Zeitpunkt eine aktuelle Liste von Subscribern erforderlich.
- **Subscriber:** Der Subscriber ist der Abnehmer für die Daten. Nach Erhalt einer Information über die Existenz des Datendienstes meldet er sich beim Dienst an und empfängt entsprechend ab dem Zeitpunkt der Anmeldung die Benachrichtigungen (Notifications) des Publishers.
- **Trust & Authentication Unit (TAU):** Die TAU erfüllt eine Doppelfunktion, die aufgrund der ähnlichen Anforderungen gemeinsam verortet werden.
    - Authentifikation: Die TAU überprüft die Authentizität (ggf. Identität) und Vertrauenswürdigkeit von Publisher und Subscriber. Diese Funktionalität wird auch als Dienst für die anderen Rollen bzw. Funktionalitäten angeboten. Für die Überprüfung können je nach Funktionsumfang der beteiligten Entitäten verschiedene Verfahren, etwa hardware- oder softwarebasierte Attestierung, zum Einsatz kommen.
    - Schlüsselserver: Die TAU stellt weiterhin für die Teilnehmer Schlüssel für den Aufbau der benötigten sicheren Kanäle (je nach Anwendung authentisch und integer oder zusätzlich auch vertraulich) zur Verfügung.
- **Authorization Policy Service (APS):** Der APS verwaltet Richtlinien für die Anmeldung zum Publikationsdienst und dient als Policy Decision Point für den Pub/Sub-Dienst. Dies kann sinnvollerweise zentral für mehrere Pub/Sub-Dienste geschehen. Die Policies können extern gesetzt werden, etwa jeweils vom entsprechenden Publication Server. Anfragen zur Anmeldung erfolgen nicht an den Publikationsserver sondern an den APS, dies ermöglicht einen zentralen Anmeldepunkt und ein Verbergen der eigentlichen Publikationsserver vor unautorisierten Kommunikatiosteilnehmern.

Je nach Design kann speziell der APS auch mit entweder einem Publication Server oder der TAU zusammen implementiert werden. Das Zusammenspiel der verschiedenen Funktionalitäten wird in Abbildung 25 beispielhaft an der Anmeldung / Registrierug eines Subscribers dargestellt.

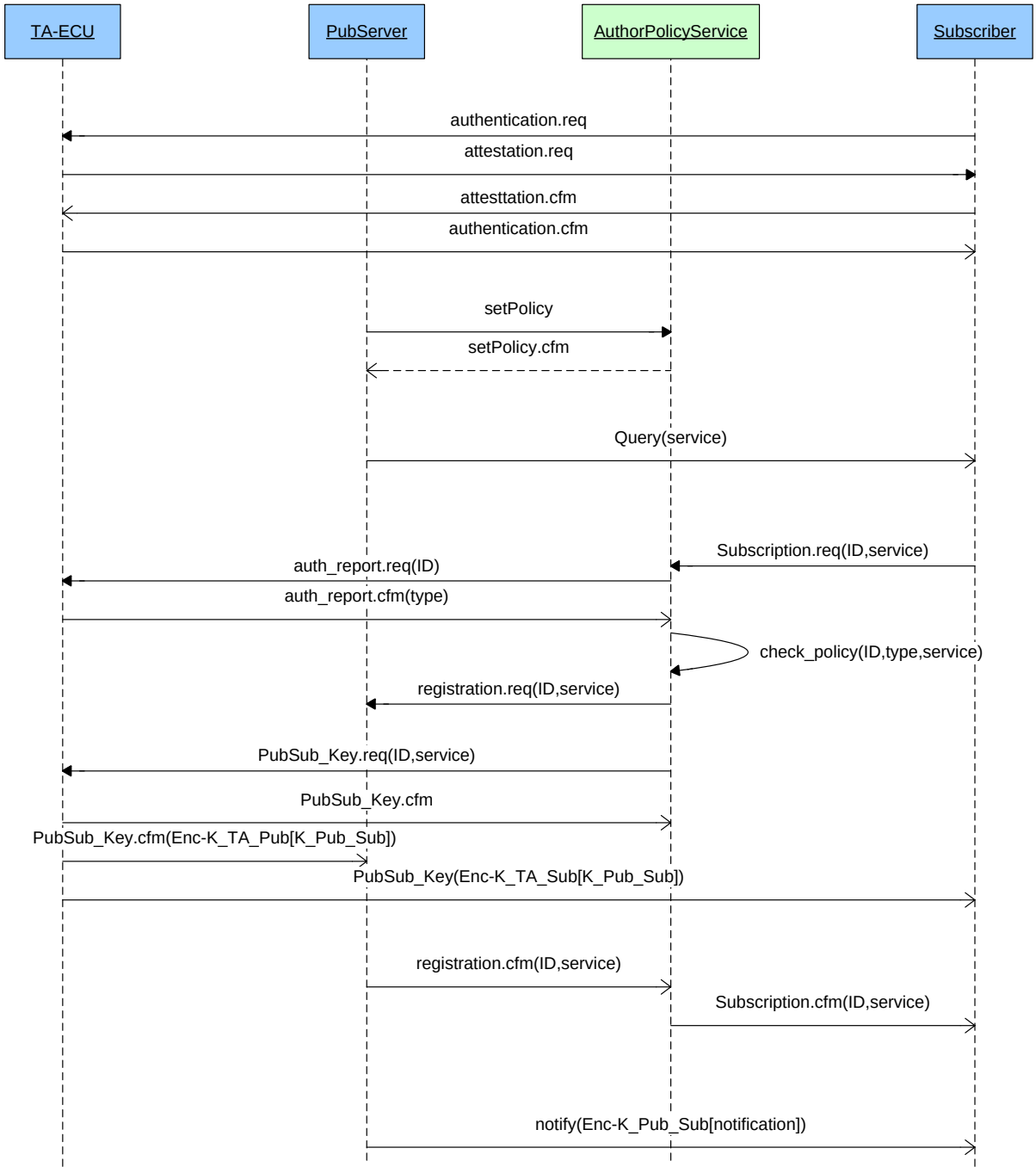

Abbildung 25: Zusammenspiel der Pub/Sub-Funktionalitäten am Beispiel der Subscription.

Zunächst erfolgt in einer Initialisierungsphase die Authentifizierung der Kommunikationsteilnehmer gegenüber der TAU, die für die direkte Kommunikation später als Trusted Third Party (TTP) fungiert. Für nach dem Aufstart des Systems später hinzukommende Teilnehmer kann die Auhentifizierung entsprechend später durchgeführt werden. Auch kann die Sicherheitsrichtlinie des Gesamtsystems eine regelmäßige Re-Authentifizierung vorschreiben. Als weiterer Schritt der Systeminitialisierung können Policies auf dem APS gesetzt werden (im vorliegenden Beispiel direkt durch den PubServer). Alternativ kann von

statischen Policies ausgegangen werden, die im APS etwa am Bandende fest gesetzt werden und deren Änderung ein externes Update des APS mit entsprechender Autorisierung erfordert.

Nach Abschluss der Initialisierung kann der Publication Server oder APS den angebotenen Dienst veröffentlichen (Query), beispielsweise in Form eines Service Announcements als lokaler Broadcast. Möchte sich ein Interessent nun bei dem angebotenen Dienst anmelden, sendet er ein Subscription Request an den APS. Dieser erfragt den Authentifizierungsstatus des Anfragers bei der TAU und entscheidet aufgrund der Antwort und den vorhandenen Policies, ob die Anmeldung autorisiert werden kann. Bei einer positiven Entscheidung fordert er entsprechende Schlüssel für PubServer und Subscriber bei der TAU an, die dann direkt zu den einzelnen Kommunikationspartnern gesendet werden. Hierbei wird von bestehenden sicheren Verbindungen der TAU zu allen authentifizierten Teilnehmern ausgegangen. Zusätzlich sendet der APS einen Registration Request für den anfragenden Subscriber an den PubServer. Nach Empfang der entsprechenden Bestätigung ist der Subscriber registriert und erhält eine Bestätigung vom APS. Ab diesem Zeitpunkt befindet er sich in der Liste der Abonnenten auf dem PubServer und erhält die angefragten Benachrichtigungen (Notifications) über den entsprechenden Kanal.

10.2.2 Externe Pub/Sub-Dienste

Die Integration externer Pub/Sub-Dienste (bspw. für den Use Case der CE-Integration) zwingt das Fahrzeug, dynamisch zusätzliche, neue externe Geräte mit unterschiedlichen Security-Fähigkeiten zu registrieren. Natürlich kann das Fahrzeug nicht alle Geräte a priori kennen, die sich verbinden wollen, und muss aufgrund einer aufgebauten Vertrauensbeziehung entscheiden, ob ein externer Dienst autorisiert ist oder nicht. Im Folgenden betrachten wir zwei Wege, externe Pub/Sub-Dienste zu integrieren: Einen direkten Weg, der direkte Kommunikation zwischen externen und internen Diensten erlaubt, und einen indirekten mit Verwendung eines Security Proxies als Vermittler.

10.2.2.1 Direkte Einbindung

In dieser Lösungsvariante wird einem externen Gerät ermöglicht, direkt auf den Kommunikationsbus zu schreiben und mit internen Pub/Sub-Diensaten zu kommunizieren. Der Proxy fungiert als transparenter Gateway und leitet den Datenstrom an interessierte ECUs weiter. Dieses Szenario erlaubt Ende-zu-Ende-Security, doch sowohl interne als auch externe Teilnehmer müssen Security-Mechanismen implementieren und ausführen, die die Security-Regeln (Policies) des internen Netzwerks umsetzen.Weitere Security-Funktionen können trotzdem auf Proxy-Ebene für die Verbindung hinzugefügt werden:

- Firewall-Infrastruktur um eingehende und ausgehende Datenströme zu filtern.
- Paketanalyse für Konsistenzüberprüfung des eingehenden Datenstroms (falls unverschlüsselt).

Allerdings hat diese Lösung einige Beschränkungen. So muss sich die auf dem externen Gerät (EG) ausgeführte Anwendung der Security-Regeln des Fahrzeug-internen Security-Protokolls bewusst sein. Das System muss eine zuverlässige Service Discovery für das EG zur Verfügung stellen, um die Teilnehmer zu identifizieren, die für Publish- und Subscribe-Funktionen zu kontaktieren sind. Und jede fahrzeuginterne ECU, die in direktem Kontakt zu einem EG steht, muss Mechanismen zur starken Authentifizierung, Autorisierung, zum Integritätsschutz, und zur Zugriffsbeschränkung bereitstellen und den entsprechenden Regelsatz (set of policies) anwenden. Die Regelverwaltung (policy management) kann daher recht komplex werden und jede beteiligte ECU benötigt ausreichend Rechenleistung für die verschiedenen Mechanismen.

10.2.2.2 Indirekte Einbindung

Im Fall einer indirekten Einbindung dürfen EGs nicht direkt mit internen Teilnehmern kommunizieren. Der Proxy empfängt die eingehenden und ausgehenden Datenströme und verfährt entsprechend seiner Security-Anforderungen und –Regeln [6]. Diese Lösung ermöglicht Punkt-zu-Punkt Security-Ansätze und der Proxy erlaubt eine Entkopplung der internen und externen Security-Mechanismen. Sie erlaubt weiter eine größere Flexibilität in der Auswahl der Protokolle – das interne System kann eine eigene Security-Lösung implementieren, unabhängig von der jeweiligen externen Lösung. Der Proxy kann basierend auf Identität und angebotenen Security-Features jeder externen Entität einen Vertrauenslevel zuordnen und damit die Information, die zwischen internen und externen Systemen ausgetauscht wird, feingranular kontrollieren.

Aus Sicht des (internen oder externen) Publishers stellt der Proxy den Subscriber dar und handelt in dessen Auftrag, umgekehrt ist wird der Proxy von Seiten des Subscribers als Publisher wahrgenommen.

Aufgrund statischer automotive-Einschränkungen und Safety-Betrachtungen erscheint die indirekte Integration adequater. Der Proxy stellt die Korrelation her zwischen dem internen System bekannten EG-Klassen und der jeweiligen EG-Identität. Die Konfiguration des internen Safety-kritischen Dienste kann statisch sein und sie müssen keine ressourcenintensiven Security-Mechanismen implementieren und ausführen. Der Dienst kann somit effizienter und in einer sichereren, kontrollierteren Umgebung betrieben werden.

10.2.2.3 Eigenschaften des Security-Proxys

Der Proxy spielt eine zentrale Rolle in der Abwehr externer Angriffe. Er stellt die erste Hürde dar für eingehende Gefahren und die letzte für ausgehende privacy-relevante Informationen. Jeder Datenstrom mit einem externen Endpunkt führt durch den Proxy. Der Security-Proxy sollte daher über genug Ressourcen verfügen, um auch umfangreiche Security-Mechanismen auszuführen, um die verschiedenen Security-Anforderungen zu erfüllen:

- Authentifizierung: Der Proxy sollte für die Authentifizierung externer Dienste verantwortlich sein.
- Integrität: Der Proxy sollte Integritätsschutz-Mechanismen bereitstellen, die unautorisierte Manipulation und Replay von Paketen erkennen.
- Vertraulichkeit: Der Proxy sollte Vertraulichkeits-Mechanismen (Verschlüsselung) zur Verfügung stellen, um den externen Informationsfluss gegen Mithören zu schützen und das Bekanntwerden Privacy-relevanter Informationen zu verhindern.
- Autorisierung: Im Gegensatz zum internen System sollte der Proxy in der Lage sein, dynamisch Security-Policies zu verändern, um neue externe Dienste zu integrieren. Die Einbindung hängt jeweils von der Evaluation des Vertrauenslevels zu dem externen Gerät bzw. Dienst ab, worauf basierend eine API-Beschränkung definiert und das Security-Risiko des entsprechenden Dienstes abgeschätzt wird. Der Vertrauenslevel sollte basierend auf den dem Proxy vorliegenden Informationen bestimmt werden. Abhängig vom Vertrauenslevel kann die Durchsetzung der Policies auf Proxy-Ebene oder auf ECU-Ebene erfolgen.

In Hinblick auf den Securityzonen-Übergang sollte der Proxy als DMZ (demilitarisierte Zone) fungieren und die erforderlichen Sicherheitsmechanismen zur Verfügung stellen, um Daten zwischen der externen Kommunikationszone und Komfortzone, bzw. zwischen der externen Kommunikationszone und Hochsicherheitszone auszutauschen. Bei Zweiterem stellt der Proxy implizit einen „Zwischenweg“ durch die Komfortzone zur Verfügung. Diese Mechanismen sollten verträglich sein mit Authentifizierung, Integritätsschutz, Vertraulichkeitsschutz und Privacy-Schutz für sowohl die Daten selbst als auch die Datenquelle.

Ein Publication-Prozess zur externen Umgebung ist ein kritischer Prozess, da der Publisher seine Nachrichten an eine Multicast-Adresse versendet und die tatsächliche Liste der endgültigen Empfänger nicht feingranular kontrollieren kann, die diese nur dem Proxy bekannt sind [7]. Der interne Publisher sollte daher der Nachricht eine allgemeine Beschreibung der erlaubten externen Empfänger (Anforderungen an externe ECUs) beifügen, aufgrund derer der Proxy diese Entscheidung durchsetzen und die entsprechenden Daten nur an konforme, autorisierte Empfänger weiterleiten kann.

10.3 SERVICE DISCOVERY

Der Service Discovery Dienst ist ein notwendiger Teil der Middleware, er stellt eine automatische Detektion von MW-Diensten (RPC und Pub/Sub) sicher. Diese SD Mechanismen erlauben:

- die Veröffentlichung von Dienstbeschreibung, Status und Location im gesamten System;
- die Abfrage von solchen Informationen;

10.3.1 Beschreibung

In einem solchem Prozess greifen 3 Aktoren ein:

- Service-User (SU): er sucht einen spezifischen Dienst, und will gesuchte Dienste identifizieren und lokalisieren;
- Service-Provider (SP): er stellt einen Dienst zur Verfügung, und will seine Dienstinformationen veröffentlichen;
- Service Directory (SDir): er listet alle verfügbaren Dienste des Systems (Beschreibung, Status und Location) auf, und bietet diese Informationen nachfragenden SUs an;

Diese Entitäten tauschen unterschiedliche Arten von Nachrichten aus, um Dienste zu veröffentlichen und zu lokalisieren (siehe Ablauf der Prozesse in Abbildung 26):

- **Service Request** (SrvRq): Nachfrage für die Beschreibung eines spezifischen Dienstes;
- **Service Reply** (SrvRp): Antwort zu SrvRq (Dienstbeschreibung);
- **Service Advertisement** (SrvAd): Dienstveröffentlichungsnachricht;
- **Service Acknowledgement** (SrvAck) Acknowledgement-Nachricht für SrvAdv;
- **Directory Advertisement** (DirAd) Nachricht für die Ankündigung von SDir;

Für SD-Mechanismen gibt es 2 mögliche Architekturen:

- „Immediate“ Architektur (ohne Service-Directory Unterstützung, siehe Abbildung 26): SP veröffentlicht die Präsenz seines Dienstes per Broadcast, SU antwortet, wenn er interessiert ist. SU kann Informationen über die Verfügbarkeit eines Dienstes via Broadcast nachfragen, der SP antwortet in Unicasting, wenn er einen solchen Dienst bietet. Eine solche Architektur ist für eine mobile Umgebung geeignet aber wenig skalierbar.
- „Mediated“ Architektur (mit Service-Directory Unterstützung, siehe Abbildung 27): SU und SP stützen sich auf einen zentralen Koordinationsdienst für die Dienstveröffentlichung (das SDir). SP und SU melden sich an dem SDir an, um SD-Information zu veröffentlichen und zu erhalten. Eine solche Architektur ist für eine mobile Umgebung kaum geeignet, skaliert dafür aber gut.

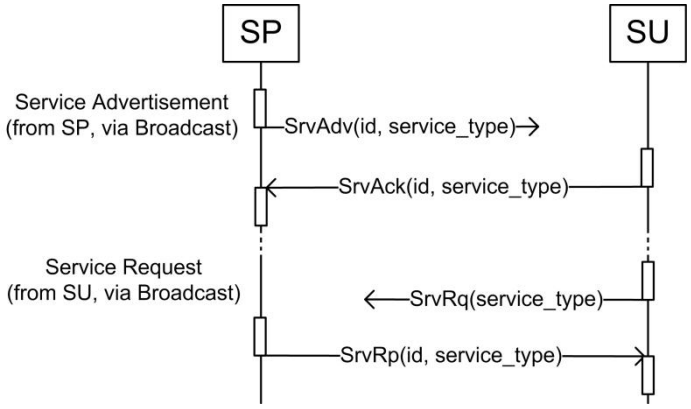

Abbildung 26: „Immediate“ Architektur

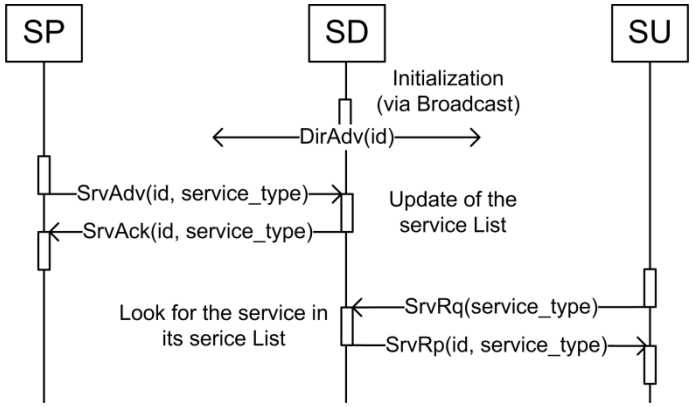

Abbildung 27: „Mediated“ Architektur

10.3.1.1 Service Discovery und Anwendungsfälle

Jede Publisher-ECU und ECU, die einen Dienst anbietet, muss Dienstinformation durch SD-Mechanismen veröffentlichen. Und Jede Subscriber-ECU und ECU, die sich zu einem RPC-Dienst anmelden will, muss die Location und den Status des Dienstes auch durch SD-Mechanismen nachfragen. Im Folgenden sind die Beispiele aus den oben aufgeführten Anwendungsfällen aufgeführt, die SD-relevant sind.

- **CE-Integration:**
  Die HU (SP) veröffentlicht ihren Dienst auf dem Service-Directory (Prozess am Bandende). Das CE-Gerät (SU) entdeckt ihn mittels der SD-Mechanismen für externe Geräte (Prozess zur Laufzeit).
- **Software-Download:**
  Der Backend-Server veröffentlicht seinen Dienst auf dem Service-Directory durch SD-Mechanismen für externe Geräte (Prozess zur Laufzeit). Die HU (SU) entdeckt den Dienst, wenn sie die Software benutzt (Prozess zur Laufzeit).
- **Remote Diagnose:**
  Die Motorsteuerungs-ECU (SP) veröffentlicht ihren Diagnose-Dienst auf dem Service-Directory (Prozess am Bandende), das Diagnosegerät (SU) findet den Dienst durch SD-Mechanismen mit dem Proxy und dem Service-Directory (Prozess zur Laufzeit).
- **Rückfahrkamera:**
  Die Rückfahrkamera ECU (SP) veröffentlicht ihren Dienst auf dem Service-Directory (Prozess am Bandende). Die HU (SU) entdeckt den Dienst durch SD-Mechanismen für interne Geräte (Prozess am Bandende).

10.3.1.2 Angriffsmodell

Wir betrachten als Angriffe jede Einwirkung, die sich gegen die SD-Ziele stellt oder in einem bösartigen Weg wirkt [18]:

- Gegen die Suche nach Diensten
- Gegen die Veröffentlichung von der Dienstbeschreibung.

Erfolgreiche Angriffe können unterschiedliche Komponenten des Dienstes kompromittieren:

- Integrität und Verfügbarkeit der Entitäten (SU, SP, SDir): Angriffe können die Entitäten in unterschiedlichen Ebenen kompromittieren (Software und Hardware), um die Kontrolle zu ergreifen und zum Beispiel falsche oder kompromittierte Dienste zum SDir und SU auszuschreiben. Außerdem kann die Kooperation zwischen SP, SU und SDir verhindert werden.
  - Angriffsbeispiele: Software-, Hardware-Angriffe.

- Vertraulichkeit, Integrität und Verfügbarkeit von den Dienstnachrichten: Angriffe können auf den Kommunikationsmedien durchgeführt werden, um den Nachrichtaustausch zu kompromittieren.
    - Angriffsbeispiele: Spoofing (Masquerading), Protocol-Tampering (Packet-Injektion, Abhören, Wiedergabe und Veränderung)…
- Unautorisierte Verwendung/ Belegung von ECU-Ressourcen [23]: Ein Angreifer kann auch die physikalischen Ressourcen kompromittieren (Bandbreite, Konnektivität, Strom).
    - Angriffsbeispiele: DoS-Angriffe (Flooding), Signal-Jamming (Bus-System, und drahtlose Kommunikation), Stromabschaltung…

Erfolgreiche Angriffe haben unterschiedliche Auswirkung auf das System:

- Kompromittierung eines SU: Die Auswirkung auf das System hängt von der Wichtigkeit des betroffenen Dienstes ab (von begrenzt bis kritisch).
- Kompromittierung eines SP: Die Auswirkung auf dem System hängt von der Wichtigkeit des betroffenen Dienstes (von begrenzt bis kritisch).
- Kompromittierung eines SDir: Die Auswirkung auf das System ist kritisch. Der ganze SD-Prozess ist kompromittiert. Die Kompromittierung muss detektiert werden und SU und SP müssen den SD-Prozess zwischen sich selbst implementieren können (gemischte Architektur zwischen „Mediated“ und „Immediate“).

10.3.1.3 Sicherheitsanforderungen

- **Authentifizierung**: Beidseitige Authentifizierung zwischen SU/SDir, SP/SDir und SP/SU ist erforderlich.
- **Autorisierung**: SU benötigt nur eine begrenzte Sicht auf die verfügbaren Dienste von SP und SDir. Nur autorisierte SP und SDir dürfen ihre Dienste ausschreiben.
- **Integrität**: SD-Nachrichten, die während der Vermittlung verändert wurden, müssen detektiert werden (z.B. Änderung, Löschen, Einfügung, Wiedergabe).
- **Nichtabstreitbarkeit**: Eine Aktivität im SD-Prozess darf später nicht geleugnet werden.
- **Verfügbarkeit**: Geeignete Maßnahmen müssen implementiert werden, um DoS-Angriffe zu begrenzen (Anwendungsfall abhängig).
- **Privatsphäre**: Dienstinformation dürfen nicht-autorisierten Entitäten nicht bekannt werden. Externe Entitäten dürfen nur nicht sicherheitskritische Informationen von SD-Protokollen bekommen.

10.3.2 Sicherheit und Service Discovery in einer IP-basierten Middleware

Die Ziele des Service-Discovery Dienstes wurde im AP 3.1 definiert. Er muss mehrere Möglichkeiten bieten:

- Für Dienstanmeldung: Service-Provider können ihren Dienst für das ganze System veröffentlichen oder nur auf einem Service-Directory anmelden (und löschen), und den Status dieses Dienstes veröffentlichen und verändern (aktiv oder inaktiv).
- Für Dienstnachfrage: Der Service-User ist fähig, ein Dienst an einem Service-Provider oder im ganzen System nachzufragen, und kann den Service-Provider anfragen, den Status seines Dienstes zu übermitteln oder zu verändern.

Der SD-Dienst von SEIS ist eine gemischte Architektur, zwischen „Immediate“ und „Mediated“. In den meisten Fällen wird eine Lösung mit Service-Directory bevorzugt und verwendet, aber für den Fall eines Fehlers dieser Entität müssen die Service-User und Provider fähig sein, die direkte Lösung zu implementieren.

10.3.2.1 Interner sicherer SD-Dienst und „Mediated“ Architektur

In diesem Prozess kennen SP und SU die Adresse von SDir, und können somit einen direkten sicheren Kommunikationskanal aufbauen. Dies betrifft jede Nachricht in Richtung SDir für Dienstregistrierung, Dienstnachfrage und Dienststatusveränderung.

Der Ablauf unterteilt sich in 3 Schritten (siehe Abbildung 28):

- Authentifizierung/Attestierung und Schlüsselaufstellung: diese Sequenz hängt vom Security-Protokoll (IPsec, SSL/TLS, Preshared-Key, User/Passwort, SSO-Lösung) ab.
- Die Nachfrage: Diese Nachricht ist verschlüsselt und die Integrität der Daten ist gesichert. (die Nachricht kann auch signiert werden, ggf. PKI-Infrastruktur)
- Die Empfangsbestätigung: Diese Nachricht ist verschlüsselt und die Integrität der Daten ist gesichert. (die Nachricht kann auch signiert werden, ggf. PKI-Infrastruktur)

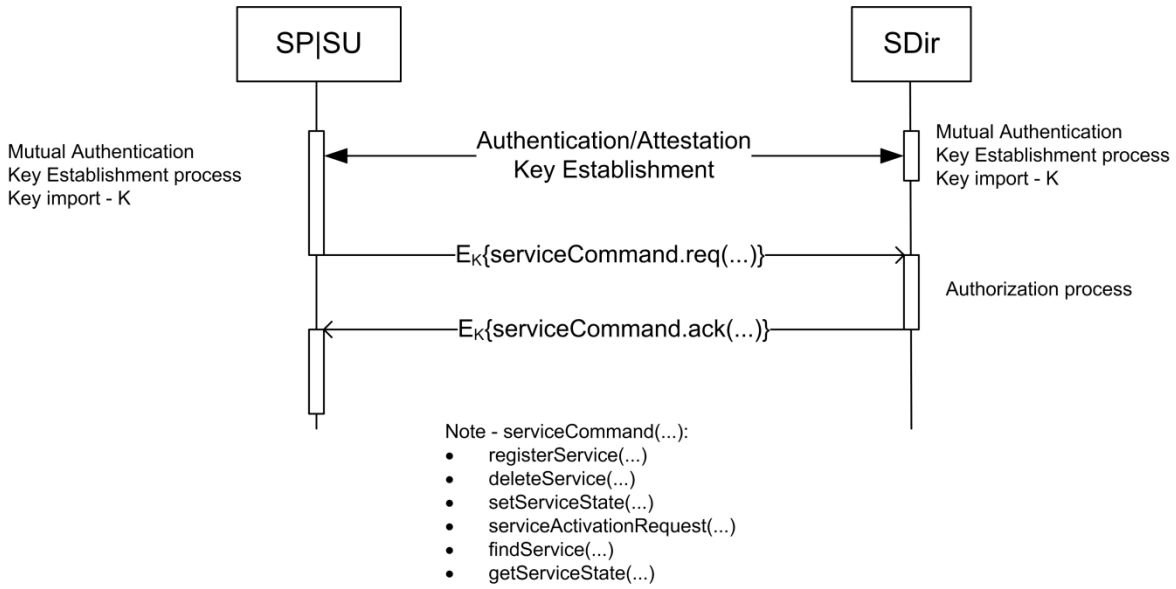

Abbildung 28: SD Ablauf für "Mediated" Architektur

Für internen Dienst erfolgt die Dienstregistrierung und Dienstnachfrage vor der Laufzeit, am Bandende oder während der Updates. Allerdings erfolgt die Dienstaktivierung und Statusveränderung während der Laufzeit.

10.3.2.2 Interner sicherer SD-Dienst und „Immediate“ Architektur

In diesem Prozess kennen SP und SU die Adresse des anderen nicht, und können somit keinen direkten Kommunikationskanal aufbauen. Dies betrifft Nachrichten für Dienstankündigung, Dienstnachfrage und Dienststatusankündigung.

Der Ablauf erfolgt in 3 Schritten (siehe Abbildung 29):

- Die Nachfrage: Diese Nachricht ist signiert, die Daten-Integrität ist geschützt. Die Entität kann sie via Broadcast oder Multicast schicken. Das System kann eine PKI Infrastruktur, für das Zertifikate-Management benutzen.
- Authentifizierung/Attestierung und Schlüsselaufstellung: diese Sequenz hängt vom Security-Protokoll ab (IPsec, SSL/TLS, Preshared-Key, User/Passwort, SSO-Lösung).
- Die Empfangsbestätigung: Diese Nachricht ist verschlüsselt und die Integrität der Daten ist gesichert. (die Nachricht kann auch signiert werden, PKI-Infrastruktur)

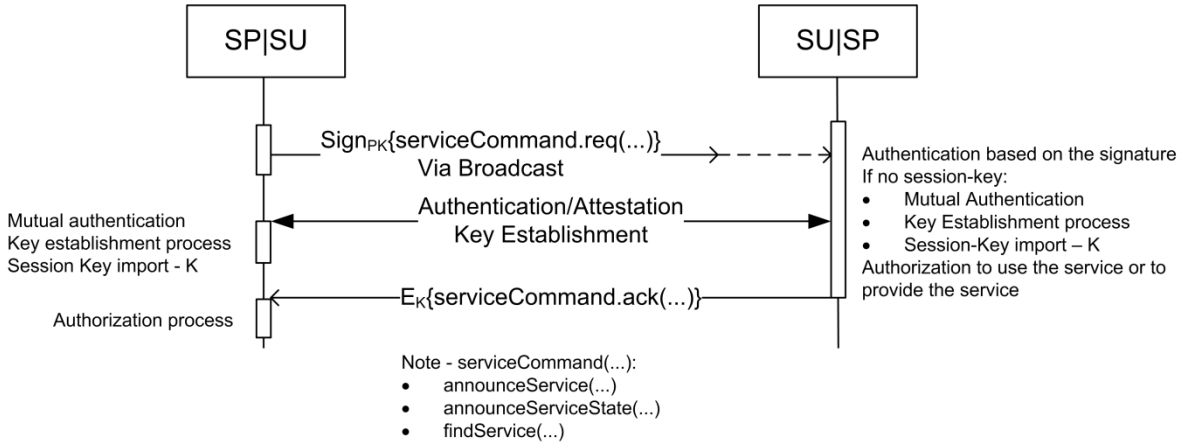

Abbildung 29: SD Ablauf für „Immediate“ Architektur

Für internen Dienst wird diese Dienstregistrierung und Dienstnachfrage verwendet, wenn die Lösung mit „Mediated“ Architektur nicht funktioniert oder nicht geeignet ist.

10.3.2.3 Externer sicherer SD-Dienst

Während der Laufzeit kann ein externes Gerät den SD-Dienst benutzen, um sich an einem internen Dienst anzumelden oder seinen eigenen Dienst auf dem Dienstverzeichnis (resp. via Broadcast) zu veröffentlichen. Bei diesem Prozess geht jede Nachricht durch den Proxy, der Proxy analysiert die Nachfrage, und leitet die Nachricht im Auftrag des externen Geräts weiter. Der Proxy informiert das Ziel der Nachricht (SDir oder SP) über die Sicherheitsebene des externen Gerätes (Set von sicherheitsrelevanten Informationen, ID, Authentifizierungs-, Verschlüsselungs-, Integritätsmethoden). Aus der Sicht des SDirs ist der Proxy ein SU, und aus der Sicht des externen Gerätes ist der Proxy ein SP. Umgekehrt, wenn eine interne ECU einen externen Dienst benutzen will, spielt der Proxy die Rolle eines SP für das interne Gerät und die Rolle eines SU für das externe Gerät. Ein solcher Prozess erlaubt es dem Proxy, geeignete Filter-Regeln und Sicherheitsmechanismen einzusetzen.

Der Ablauf gleidert sich in 4 Schritten ab (siehe Abbildung 30):

- Authentifizierungsprozess: Beidseitige Authentifizierung zwischen externem Gerät und Proxy und Aushandlung eines Kommunikationsschlüssels.
- Die Nachfrage: Diese Nachricht ist verschlüsselt und die Integrität der Daten ist gesichert. (die Nachricht kann auch signiert werden, PKI-Infrastruktur).
- Interner Prozess: Der Proxy analysiert die Nachricht und bestätigt mit einem Access-Control System, wenn die Nachricht zum internen System weitergeleitet werden kann. Für eine bessere Zugriffskontrolle auf der ECU kann der Proxy andere sicherheitsrelevante Informationen zur internen ECU weiterleiten.
- Die Empfangsbestätigung: Diese Nachricht ist verschlüsselt und die Integrität der Daten ist gesichert. (die Nachricht kann auch signiert werden, PKI-Infrastruktur), es ist die Antwort des internen Geräts.

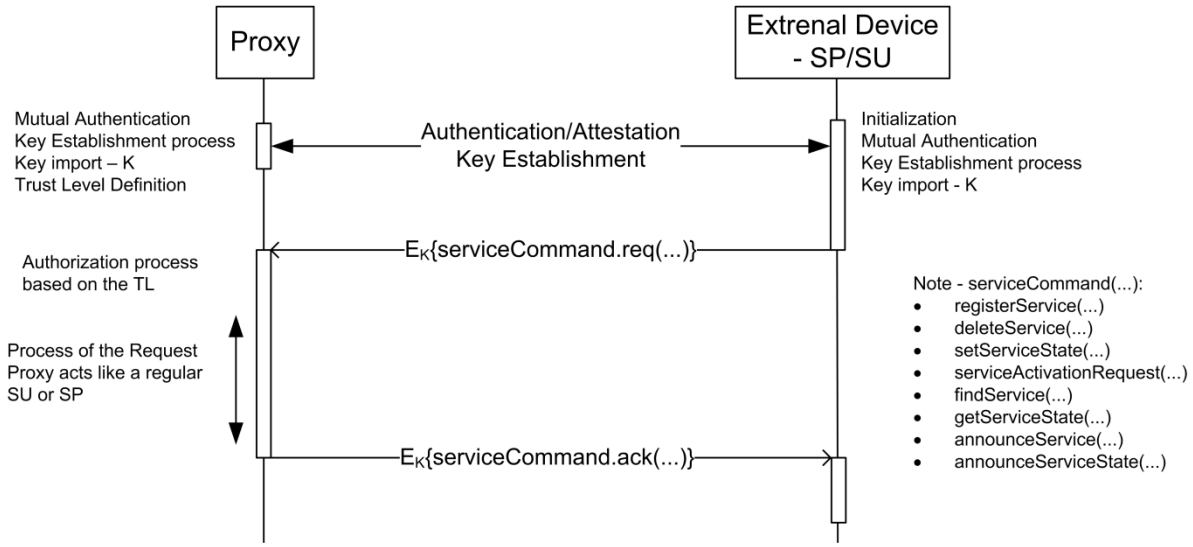

Abbildung 30: SD Ablauf mit Außenwelt

11. FEHLERMANAGEMENT INNERHALB DER SECURITY MIDDLEWARE

Im folgenden Abschnitt wird auf die Thematiken des Fehlermanagements und der Ausfallsicherheit der von SEIS entwickelten sicherheitsrelevanten Architektur eingegangen. Ein besonderes Augenmerk wird hier auf die Wichtigkeit von Fehlererkennung gesetzt.

Zuerst werden verschiedene Arten von Fehlern und Fehlerquellen betrachtet. Anschließend wird deren Erkennung und mögliche Reaktionen betrachtet. Abschließend werden zwei verschiedene Lösungsarchitekturen vorgeschlagen.

11.1 FEHLER

Fehler können in typischen Computersystemen verschiedenste Gründe haben. Einige der offensichtlichen Fehlerquellen stehen jedoch nicht im direkten Bezug zur Implementierung von Sicherheit in IP-basierten Netzwerken. Im Folgenden werden die erwarteten Fehlerquellen kategorisiert und entschieden, ob deren Lösung innerhalb von SEIS adressiert werden soll. Sollte keine Lösung innerhalb von SEIS gefunden werden, so sollte entschieden werden, ob diese Fehler im implementierten System relevant sind.

11.1.1 Bugs

Der vermutlich häufigste Grund für einen Fehler ist ein Bug innerhalb der Software oder der Hardware des neu implementierten Systems. Diese Art von Fehlern ist unabhängig von Konzepten der sicheren Verwendung von IP innerhalb von Fahrzeugen. Diese Fehlerquelle kann - und wird bereits - durch verschiedene Techniken des Software Engineerings behandelt, daher wird diese Fehlerquelle innerhalb von SEIS nicht gesondert betrachtet.

11.1.2 Defekte

Eine weitere, unvermeidbare Fehlerquelle ist der physikalische Defekt von Steuergeräten. Diese Fehlerquelle wird von SEIS nicht direkt betrachtet. Es gibt in typischen Fahrzeugen bereits mehrere einzelne Fehlerstellen (Single-Point-of-Failure). Ein Stichwort in diesem Zusammenhang ist „Mean Time Between Failure“ (MTBF). Die durch Defekt verursachten Fehler können durch eine redundante Auslegung von Steuergeräten und Diensten vermieden werden.

11.1.3 Denial-of-Service Angriffe

Angriffe auf das System sind eine weitere Fehlerquelle innerhalb von IT Systemen. Im Folgenden sollen einige Arten von Denial-of-Service Angriffen (DoS) bewertet werden.

11.1.3.1 Flooding

Bei einem Denial-of-Service Angriff geht es dem Angreifer entweder darum, das Opfer des Angriffs so mit Anfragen zu überlasten, dass es seiner eigentlichen Aufgabe nicht mehr nachkommen kann oder das Opfer mit einem präparierten Paket abzuschalten. Ein frühes Beispiel für ein solches präpariertes Paket ist der „Ping of Death“, bei dem eine Lücke im TCP/IP Stack ausgenutzt wurde um ein entferntes System zum Stillstand zu bringen. Eine mögliche Lösung des Problems ist es, Protokolle so zu entwerfen, dass der Initiator einer Anfrage mehr Rechenaufwand zu investieren hat, als der Server, welcher die Anfrage beantwortet. Aufgrund der eingebetteten und rechenschwachen Steuergeräte, welche im Fahrzeug verbaut sind, ist dies schwer umsetzbar. Daher ist die Möglichkeit eines DoS Angriffs gegen ein einzelnes Steuergerät nur schwer einzuschränken. Zusätzlich ist zu bemerken, dass eine redundante Auslegung von Steuergeräten nicht gegen DoS Angriffe schützen kann, da der Rechenaufwand für einen Angreifer nur linear steigt.

Ein Intrusion Prevention System kann bei diesem Angriff mindernd eingreifen indem es die Quelle des Angriffes identifiziert und im Netz isoliert.

11.1.3.2 Physikalische Angriffe

Ein weiterer Angriff ist zum Beispiel das physikalische Abstecken oder Verändern eines Steuergerätes. Darüberhinaus kann das Fahrzeug in einer Weise modifiziert werden, dass alle Anfragen an ein Steuergerät abgefangen und von einem vom Angreifer eingebrachten Steuergerät beantwortet werden (sog. Identitätsspoofing). Dies ist nicht auf Protokollebene innerhalb der Middleware behebbar. Zudem bietet eine redundante Auslegung der Hardware hierbei keinen zusätzlichen Schutz. Daher wird der Schutz gegen physikalsche Angriffe innerhalb von SEIS nicht betrachtet. Es besteht Konsens, dass ein effektiver Schutz vor physikalischen Angriffen nicht möglich ist. Dies soll vom folgendem Beispiel illustriert werden: Ein reiner Man-In-The-Middle Angriff kann durch die Verwendung eines Schlüsselaustauschverfahrens (z.B Diffie Hellman) verhindert werden. Hat der Angreifer jedoch zusätzlich die Möglichkeit, den Speicher eines der beteiligten Steuergeräte auszulesen, so können Authentifizierungsschlüssel entnommen und der Verkehr der beiden verschlüsselt kommunizierenden Steuergeräte ausgelesen und sogar verändert werden. Ein einzelner Angriff ist verhinderbar, eine Kombination aus verschiedenen Angriffen ist jedoch schwer zu verhindern.

11.1.4 Angriffe auf Protokollebene

Fuzzing beschreibt eine Technik, bei der zufällige Daten an ein Steuergerät oder eine Applikation gesendet werden. Dies zielt darauf ab Sicherheitslücken, wie z.B. Pufferüberläufe, in den verwendeten Protokollen oder den verwendeten Kommnikationsstacks auszunutzen. Die innerhalb des sicheren Fahrzeugnetzes verwendeten Protokolle und Applikationen müssen solche bösartigen Anfragen erkennen und gegebenenfalls verwerfen. Dies wird durch Techniken wie „Input Validation“ und Konsistenzüberprüfungen gewährleistet, welche bereits durch zertifizierte Software-Engineering-Prozesse vorausgesetzt sein sollten.

11.1.5 Skalierung des Gesamtsystems

Das von SEIS entworfene System hat Anforderungen an die Skalierbarkeit des Gesamtsystems. Diese bezieht sich sowohl auf die Anzahl der kommunizierenden Steuergeräte als auch die Erweiterbarkeit des Systems aufgrund zukünftiger Anforderungen. Deshalb müssen bei den verwendeten Systemen und Protokollen Fragen der Skalierbarkeit betrachtet werden. Innerhalb von SEIS ist dies durch den Vorschlag eines dezentralen, modularen Systems mit einem Security Framework und die Verwendung von Plug-Ins berücksichtigt.

11.1.6 Paketverluste und Nichterreichbarkeit

Bei der Kommunikation in IP-basieren Netzen ist der Verlust von einzelnen IP-Paketen nicht ungewöhnlich. Dies wird unter anderem vom IP Protokoll selbst dazu verwendet, den Kommunikationspartnern eine Sättigung der unterliegenden physikalischen Kanäle zu signalisieren. Derartige Paketverluste werden bei Verwendung von TCP von der Transportschicht erkannt. Sollte ein anderes Protokoll der Transportschicht verwendet werden, müssen Paketverluste eventuell speziell erkannt und behandelt werden.

Ein weiteres Problem ist die Nichterreichbarkeit von Kommunikationspartnern. Kurzfristige Nichterreichbarkeit kann unter Umständen mit Timeouts behandelt werden. Damit die Sitzungsdaten darüber hinaus nicht aufwendig neu ausgehandelt werden müssen, können Techniken wie „Dead Peer Detection“ oder „Session Resumption“ zum Einsatz kommen. Längerfristige Nichterreichbarkeit (die nicht auf bewusste (Teil-)netzabschaltung zurückzuführen ist) ist jedoch auf physikalischen Defekt oder eine erfolgreiche DoS-Attacke zurückzuführen. Diese wird innerhalb von SEIS nicht betrachtet.

Was jedoch innerhalb von SEIS innerhalb von AP4.4 betrachtet wird, ist eine dauerhafte Nichterreichbarkeit eines Dienstes, der auf einem Gerät betrieben wird, welches mehrere Identitäten im System besitzt (Mobility und Multihoming). Die Verwendung von Mobility und Multihoming ist jedoch innnerhalb des Fahrzeugnetzes nicht empfehlenswert. Hier sollte eine statische Konfiguration bevorzugt werden.

11.1.7 Fehler bei der Authentifizierung

Ein weitere Fehlerquelle innerhalb des Systems sind Fehler bei der Authentifizierung oder innerhalb des Policymanagements. Diese sind unter anderem: Fehlende Berechtigung eines Steuergeräts, abgelaufene Schlüssel oder Zertifikate (sowohl auf Client, als auch auf Server Seite) oder zu lange Reaktionszeiten bei der Attestation. Mögliche Ursachen für diese Fehler sind unter anderem eine fehlgeschlagene Attestation, Revokation des Schlüsselmaterials oder Paketverluste während der Attestation. Diese Fehler sind aufgrund der verwendeten Sicherheitsprotokolle unvermeidbar und müssen deshalb innerhalb des Security Frameworks behandelt werden.

11.1.8 Zusammenfassung der Fehlerfälle

Tabelle 18 bietet eine Übersicht über die vorgestellten Fehlerursachen.

Nicht betrachtete Fehlerquellen	Betrachtete Fehlerquellen
Bugs	Angriffe auf Protokollebene
Flooding	Skalierung des Gesamtsystems
Physikalische Defekte	Paketverluste / Nichterreichbarkeit
Physikalische Angriffe	Fehler der Middleware

Tabelle 18: Fehlerquellen

11.2 FEHLERERKENNUNG

Im Folgenden wird betrachtet, wie die einzelnen, im vorhergehenden Abschnitt beschriebenen Fehler zuverlässig erkannt werden können.

11.2.1 Intrusion Detection

Ein zur Erkennung von externen Einflüssen und insbesondere Angriffen eingesetzte Technik ist im Allgemeinen ein Intrusion Detection System (IDS). Bei der Implementierung eines IDS stößt man unweigerlich auf „False Positives“, sowie „False Negatives“, man erkennt also vermeintlich Angriffe, die keine sind und erkennt unter Umständen tatsächliche Angriffe nicht. Daher ist die Entwicklung und Konfiguration eines IDS ein wichtiges Thema. Als eine weitere Fragestellung ergibt sich in diesem Zusammenhang, in wieweit die Implementierung eines IDS ohne einen dedizierten Trust Anchor sinnvoll ist. Hier könnten eventuell Introspection-basierte Verfahren zur Anwendung kommen. Das Thema Intrusion Detection wird innerhalb von SEIS in AP4.2 behandelt. Innerhalb von AP4.3 wird Intrusion Detection innerhalb des Intrusion Management Moduls der Middleware betrachtet.

11.2.2 Unterschied zwischen Angriff und Defekt

Ein zusätzliches Problem bei der Erkennung von Fehlern ergibt sich bei der Unterscheidung zwischen Angriffen und Defekten innerhalb des überwachten Systems, da der für das System erkennbare Effekt unter Umständen der gleiche sein kann. Dies soll folgendes Beispiel illustrieren: Das ABS System ist für die Fahrsicherheit relevant. Ist das ensprechende Steuergerät defekt oder kompromittiert, ist die Sicherheit der Fahrzeuginsassen gefärdet. Mit Hilfe von Intrusion Management kann das ABS-Steuergerät melden, dass es defekt ist oder kompromittiert wurde. Je nach Defekt (oder Kompromittierung) kann diese Meldung jedoch eventuell nicht mehr versendet werden. Dies soll illustrieren, dass sich der Rest des Fahrzeugs nicht auf aktive Meldungen über einen Defekt verlassen kann. Andererseits kann es je nach Kompromittierung vorkommen, dass ein Angreifer in der Lage ist, eine Defektmeldung zu senden. Auch hier kann sich das Fahrzeug nicht auf die Fehlermeldungen verlassen. Darüber hinaus kann ein

Angreifer durchaus einen Defekt simulieren und dadurch als Angriff nutzen. Daraus ergibt sich, dass Defekte auch als Angriffe auf die Fahrzeuginfrastruktur gewertet werden müssen.

11.3 Verhalten im Fehlerfall

Neben den möglichen Fehlern und der Fehlererkennung muss das Verhalten des Systems im Fehlerfall betrachtet werden. Hierbei ist zu beachten, dass ein Szenario mit implementierter Sicherheit im Fehlerfall nicht schlechter sein darf, als das ursprüngliche, ohne Sicherheit implementierte System. Wie im vorherigen Abschnitt klargestellt muss darüberhinaus mit Fehlern durch Defekte und Angriffe gerechnet werden. Das heißt, dass die Reaktion des Fahrzeugs auf Angriffe in keinem Fall die Fahrzeugsicherheit (Safety) beeinträchtigen darf. Dies ist insbesondere bei einem entfernten Eingreifen des OEMs ins Fahrzeug zu betrachten. Im Folgenden werden die drei Möglichkeiten: Keine Beeinträchtigung, Notbetrieb und Abschaltung betrachtet.

- Keine Beeinträchtigung
  Im Idealfall, ergibt sich aus dem Ausfall eines Teilsystems keine Beeinträchtigung für den Nutzer. Im Fehlerfall bei einem zentralen security-relevanten System, könnten z.B. safety-kritische ECUs unverschlüsselt weiter miteinander kommunizieren. Ein Abschalten von Zusatzfunktionen kann als Angriff gegen die Fahrzeugflotte genutzt werden und ist daher als kritisch einzustufen.
- Notbetrieb
  Da ungesicherte Kommunikation eine große Gefahr darstellen kann, könnten im Fehlerfall Funktionen, welche nur durch die mögliche Verschlüsselung im Fahrzeug implementiert wurden, wie z.B. das Anzeigen von Navigationsmaterial, deaktiviert werden. Safety-kritische Dienste müssen jedoch weiter zur Verfügung gestellt werden. Es ist z.B. daran zu denken, die Maximalgeschwindigkeit des Fahrzeugs zu reduzieren, oder bestimmte Warnsignale auszugeben.
- Abschaltung
  Eine weitere Möglichkeit der Fehlerbehandlung ist die sofortige Notabschaltung des Systems. Diese ist äußerst vorsichtig zu implementieren, da sich hieraus weitere Implikationen auf die Fahrsicherheit (Safety) ergeben. Eine abgemilderte Möglichkeit ist, ein ausgeschaltetes Fahrzeug bei Fehlern nicht wieder zu starten.

11.4 Mögliche Architekturen

Für SEIS ergeben sich aus den aufgezeigten Fehlerfällen zwei mögliche Architekturen: eine zentralisierte Architektur mit zentralem Vertrauensanker und zentraler Policy Verwaltung oder eine komplett dezentrale Architektur.

11.4.1 Zentral verwaltete Architektur

Der erste Vorschlag ist eine zentrale verwaltete Architektur mit einem zentralen Server, der bestimmte Dienste zur Verfügung stellt, und beliebig vielen Steuergeräten (Clients), welche die Dienste des Servers in Anspruch nehmen. Daraus ergibt sich, dass der zentrale Server einen Single-Point-of-Failure (SPOF) darstellt. Sollte der zentrale Server aus einem beliebigen Grund ausfallen, so ist das gesamte System nicht mehr voll funktionsfähig.

- Verhinderung eines Single-Point-of-Failures
  Ein SPOF kann nur dadurch vermieden werden, dass es keine einzelne zentrale Instanz zur Verwaltung der Security im System gibt. Im Idealfall sollte jede ECU dies selbstständig erledigen können. Der Wegfall der zentralen vertrauten Instanzen bedingt jedoch die Einführung von vielen lokalen Vertrauensankern in Form von Hardware Security Modulen (z.B HSMs oder TPMs).
- Vermeidung eines Single-Point-of-Failures
  Eine Architektur zur Vermeidung von SPOF kann einerseits eine redundante Architektur mit kaskadierten Zuständigkeitsbereichen oder eine Primary/Slave Architektur mit „Heartbeat“ sein. Andererseits kann ein System verwendet werden, welches auf Techniken wie P2P oder Voting-

Verfahren aufsetzt. Das redundante System besitzt eine geringere Ausfallwarscheinlichkeit. Deshalb ist eine redundante Auslegung des Systems vorzuziehen.

Da ein Single-Point-of-Failure im Fahrzeug nicht verhinderbar ist, wird empfohlen auf eine redundante Sicherheitsarchitektur zurückzugreifen.

11.4.2 Dezentrale Architektur

Bei einer dezentralen Architektur werden alle Dienste, insbesondere die des Security Frameworks, dezentral angeboten. Sie hat den Vorteil, dass sie durch mehr Dynamik mehr Sicherheit hinsichtlich Ausfallsicherheit (Safety) und Angriffssicherheit (Security) bietet. Diesem Vorteil steht jedoch ein höherer Wartungsaufwand, sowie eventuell höhere Kosten für mehrere TPMs entgegen. Es ist also abzuwägen, ob die Gefahr eines SPOF den Aufwand einer dezentralen Architektur rechtfertigt.

12. ZUSAMMENFASSUNG UND AUSBLICK

Dieses Ergebnisdokument stellt die SEIS Sicherheitsarchitektur für eine IP-basierte Fahrzeug-Middleware vor. Es enthält eine ausführliche Beschreibung der Software-Security-Module. Das Dokument schlägt ein Security-Framework für eine flexible und optimierte Verwendung von Sicherheits-Funktionalitäten und Fahrzeug-Middleware-Diensten vor. Das Design dieser Architektur wurde basierend auf den Sicherheitsanforderungen vom AP1.3 [16] erstellt. Das Sicherheits-Framework wurde angelegt, um mit den MW-Spezifikationen vorgestellt im AP3.1 [31] und dem Sicherheitszonenmodell vom AP4.1 [17] kompatibel zu sein. Das vorgeschlagene Framework definiert die Infrastruktur, um die im AP4.2 [14] betrachtete sichere Kommunikation zu erlauben. Dieses Dokument gibt verschiedene Versionen für alle Module und Plug-Ins an, um genügend Flexibilität für jeden Anwendungsfall zu bieten und alle notwendigen Anforderungen zu erfüllen. Weitere Analysen und formale Prüfungen für die Integration in das Gesamtsystem werden im AP4.5 [21] behandelt.

Als erstes Ziel beschreibt dieses Dokument die Sicherheitsspezifikation im Bezug auf Middleware-Dienste für interne und externe Kommunikation. Für jeden der Dienste wurden Angreifer definiert und mögliche Angriffe und ihre Auswirkungen auf das System aufgelistet. Anschließend wurden basierend darauf die enstprechenden Sicherheitsanforderungen definiert und erste Spezifikationen zu ihrer Realisierung vorgeschlagen. Die drei betrachteten Middleware-Dienste sind die Folgenden.

- Remote Procedure Call (RPC): Dieser Dienst ermöglicht eine Kommunikation (synchrone oder asynchrone) zwischen zwei Anwendungen unabhängig von ihrem Implementierungsort;
- Publish/Subscribe (Pub/Sub): Ein zweiter wesentlicher Kommunikationsdienst, er ermöglicht „Publishern“ (Datenquellen), Daten oder Events an eine große Anzahl von „Subscribern“ (Interessenten / Datensenken) zu verteilen, die mittels einer Anmeldung zum entsprechenden Publikationsdienst ihr Interesse angezeigt haben;
- Service Discovery: Dieser Dienst ermöglicht eine gegenseitige (dynamische) Dienstentdeckung.

Als zweites Ergebnis wurde ein Security-Framework definiert, das als Grundlage für den Aufbau sicherer Kanäle und einer sicheren Laufzeitumgebung für Applikationen im Fahrzeug verwendet wird. Unsere Architektur bietet Authentifizierung, Zugriffskontrolle, sichere Kommunikation, Schlüssel-Management, Intrusion-Detection und kryptographische Mechanismen für die Anwendungen und ihre Kommunikation untereinander. Für jedes Security-Modul wurden die Auswirkungen von Sicherheitsverletzungen oder Fehlerfällen auf die Sicherheitskomponenten des Fahrzeuges evaluiert und eine Sicherheits-API definiert. Zusätzlich wurden notwendige Sicherheits-Plug-Ins und ihre Entwicklungsspezifikationen zusammengestellt. In den folgenden Tabellen fassen wir die Liste der Security-Module, ihre Zwecke und ihre zugehörigen Plug-ins zusammen.

Authentication-Management-Modul: Das AMM ist zuständig für die Anmeldungsprozedur während des Aufbaus von Kommunikationskanälen.

Plug-in	Ausprägung	Beschreibung
Internal Service Authentication Manager (ISAM)	auf einem zentralen Steuergerät oder verteilt zwischen mehreren	Authentifizierungsverwaltung für interne Dienste (Status der Authentifizierung, Methoden, Policy-Generation)
External Service Authentication Manager (ESAM)	auf dem Proxy	Authentifizierungsverwaltung für externe Dienste (Status der Authentifizierung, Methoden, Policy-Generation) - Registrierung von externen Geräten - Mapping der internen ID und richtigen ID Risikoevaluierung / Unterstützung für Policy-Enforcement

Plug-in	Ausprägung	Beschreibung
Secure Credential Maker	auf einem zentralen Steuergerät oder verteilt zwischen mehreren	Sicherheitsmaterialen-Management: Nachfrage für Erzeugung von Sicherheitsmaterialen (Schlüsseln, Signatur, HMAC…) basierend auf den Policies von ISAM und ESAM (Generation und Speicherung behandelt vom KMM und HSM)
Secure Credential Verifier	auf einem zentralen Steuergerät oder verteilt zwischen mehreren	Sicherheitsmaterialen-Management: Nachfrage für Überprüfung von Sicherheitsmaterialen (Überprüfung behandelt vom HSM) und Verwaltung (Ergebnis-Logging für IMM)
Pseudonym Manager	auf dem Proxy	Generierung des Pseudonyms; Kommunikation mit externen Infrastrukturen für die Veröffentlichung der Pseudonymen;

Key-Management-Modul: Das KMM verwaltet die kryptografischen Schlüssel des Fahrzeugsystems.

Plug-in	Ausprägung	Beschreibung
Pre-Shared Keys (symmetric keys)	Zentrale (mit einem Master) / Multi-zentriert (mit Domain-Master) / Distributed (Master-Hierarchie)	Management von PSK (Erstellung, Verteilung, Rekeying-Prozess, Speicherung)
Asymmetric Keys with PKI	Zentral/ Multi-zentriert / Verteilt	Management der digitalen Zertifikate (Erstellung, Verteilung, Widerruf und Speicherung)
Platform Attestation	Zentral/ Multi-zentriert/	Direct Anonymous Attestation Mechanismen

Policy-Management-Modul: Dieses Modul kümmert sich um das Policy-Management und den Entscheidungsprozess.

Plug-in	Ausprägung	Beschreibung
Policy Manager	Auf einem zentralen Steuergerät oder verteilt auf mehreren	Policy-Datenbank Management (für PDP): - Einfügung von Policies in geeigneten Datenbank (Aktualisierung) - Schnittstelle mit der Außenwelt für OEMs Policy-Aktualisierung Policy-Entdeckungsmethoden (für verteilte PDP)
Policy Decision Manager	Auf einem zentralen Steuergerät oder verteilt auf mehreren	Policy Decision Manager (für PDP und PEP): Generiert die Policy-Entscheidung basierend auf Policy- und Kontextparametern und speichert die Policy-Entscheidung.

Plug-in	Ausprägung	Beschreibung
Policy Converter	Auf einem zentralen Steuergerät oder verteilt auf mehreren	Übersetzung von Policies in unterschiedlichen Formaten (z.B. XACML, SAML, andere Fahrzeug-Policy-Format)

Secure Channel-Modul: Dieses Modul baut sichere Kanäle zwischen zwei Diensten auf und koordiniert die Prozesse zwischen den anderen Security-Modulen.

Plug-in	Ausprägung	Beschreibung
IPsec Plug-in	Auf jedem Steuergerät	Umsetzung von Sicherheitsmechanismen (Socket + Protokolle). Filter-Generation und Durchsetzung (Policy-basiert).
Proprietary secure communication Plug-in	Auf jedem Steuergerät	Umsetzung von Sicherheitsmechanismen (Socket + Protokolle). Filter-Generation und Durchsetzung (Policy-basiert).
TLS Plug-in	Auf jedem Steuergerät	Umsetzung von Sicherheitsmechanismen (Socket + Protokolle). Filter-Generation und Durchsetzung (Policy-basiert).

Intrusion-Management-Modul: Die Intrusion-Detection und die anderen damit verbundenen Sicherheitsmaßnahmen werden von diesem Modul realisiert.

Plug-in	Ausprägung	Beschreibung
Host IDS	Auf jedem Steuergerät / auf ausgewählten Steuergeräten	System-call Monitoring (Logs-Monitoring)
Network IDS	In jedem Subnetz / in ausgewählten Subnetzen	Network Monitoring (Traffic-Monitoring)
IPS	Auf jedem Steuergerät und Subnetz / auf ausgewählten Steuergeräten und Subnetzen	Active Counter-Reactions (Policy-Erstellung, Prozesstötung, Portschließen)

Cryptographic Services Modul: Dieses Modul stellt kryptographische Funktionen des Systems zur Verfügung (Schlüsselerzeugung, Verschlüsselung / Entschlüsselung, Signatur und HMAC Erzeugung / Überprüfung ...)

Plug-in	Ausprägung	Beschreibung
Hardware plug-ins	Auf jedem Steuergerät / auf ausgewählten Steuergeräten	Diesen AP entwickelt kein neues Design für sicheres Hardware-Modul. (Für HW-Design-Spezifikation, bitte siehe EVITA-HSM [EVITA])

Das SEIS-Projekt findet nach diversen anderen Projekten statt, die Fahrzeug-Security-Lösungen entwickeln und baut auf ihren Ergebnissen implizit und explizit auf. EVITA ist eines von ihnen und bezweckt

wie SEIS das Ziel der „Absicherung von internen und externen Fahrzeugkommunikationssystemen". Die folgende Tabelle zeigt die Erweiterungen und Mehrwerte durch dieses SEIS-Arbeitspaket auf:

Themen	EVITA [30],[32]	SEIS AP4.3
Betrachte Bus-Protokolle	Traditionelle Bus-Technologien (CAN, Flex-Ray);	IP-basierte Protokollen;
Middleware-Dienstbetrachtung	Keine Betrachtung	Sicherheitsanalyse von MW-Diensten. Spezifikation von Sicherheitsmaßnahmen
Car-to-X Communication	Betrachtung in den Anwendungsfällen und in der Risikoanalyse. Interne Protokollen können mit gegenseitiger Attestierung ergänzt und begründet werden.	Betrachtung in den Anwendungsfällen und in den Dienstanalysen. Architektur des Security Proxy (Entkopplung von Sicherheitsmechanismen).
Authentifizierung	Zentrale Entität, die die SSO, den Authentifizierungsprozess und seine Verwaltung realisiert.	Zentrale Entität, die die Verwaltung des Authentifizierungsprozesses zwischen Diensten (Authentication-Policy Management, Status-Logging) realisiert. Jeder Dienst implementiert seine eigene Authentifizierungsmethoden.
Policy Management	Verteiltes System von PEP/PDP Entitäten. Verwendung von XACML-Policies.	Verteiltes System von PEP/PDP Entitäten. Definition von mehrlagigen Policies (Kriterien-Spezifikation)
Key Management	Key-Master Architektur. Implementierung eines HSM-basierten zentralen Speichermoduls.	Basierend auf Standards, jeder Steuergerät/Dienst baut seine eigene Kommunikation (z.B. IKEv2). Entwicklung von PSK- und PKI-Management-Diensten.
Aufbau von sicheren Kanälen	Entwicklung eines Sicherheitsprotokolls CTP für Bus-Technologien.	Basiert auf Standards (IPSec für interne Kommunikation und TLS für externe Kommunikation).
Sichere Hardware	Design und Entwicklung von 3 verschiedenen HSM	Nicht betrachtet.
Intrusion Detection	Log Monitoring	Log Monitoring, Traffic Monitoring, Virus-Signatur-Datenbank
Mechanismen für Plattformintegrität	HW-basierte Vertrauensanker-Mechanismen, Virtualisierungsprozess (VM, Hypervisor)	HW-basierte Vertrauensanker-Mechanismen (Software-Attestation)
Fehlerfall-Management	Nicht betrachtet	Reflexion über Fehler-Management, Komponentenredundanz, Fail-Safe Modus

Die prototypische Implementierung der Sicherheitsarchitektur in Hardware und Software wird als Teil des SEIS-Projektes geplant. Zwei Demonstratoren wurden mit Beziehung zu diesem Arbeitspacket entwickelt.

"Automotive-Authentifizierung mit zentralem Trust-Anker für sichere Software-Updates" (Fraunhofer AISEC & ESK, VW und Continental): Automotive Komponenten können sichere Kommunikationskanäle für die unterschiedlichsten Zwecke aufbauen. Aber um eine gültige gegenseitige Authentifizierung zwischen dem Auto und Server zu gewährleisten, müssen die Sicherheitsmaterialen (Credentials / Token / Schlüssel) gegen unbefugte Manipulation geschützt werden. Dieser Demonstrator implementiert einen

zentralen Vertrauensanker auf manipulationsresistenten Smart-Chips, und bietet eine geeignete Sicherheitsstufe für die Authentifizierungsmaterialen und den Aufbau von IPSec-basierten Tunneln zum Remote-Server für eine Software-Aktualisierung.

"Security Proxy-Architektur für sichere CE-Integration" (BMW): Die meisten der CE-Sicherheitslösungen fokussieren auf den Schutz der Integrität des CE-Gerätes oder der User-Privacy an. Jedoch sichern nur wenige Lösungen das System ab, das die CE-Dienste integriert (starke Geräte- und Anwendungsauthentifizierung, angepasste Zugriffskontrolle). Dieser Demonstrator implementiert die Funktionalitäten des im Dokument präsentierten Security-Proxy. Der Security-Proxy ermöglicht Standardauthentifizierungsverfahren und Security-Level-basierte Risikoanalyse für eine sichere Integration von externen Diensten (siehe Beschreibung in AP 3.2 [28]).

Obwohl dieses Dokument ein komplettes Security-Framework für IP-basierte Fahrzeug-Middleware vorschlägt, bleiben einige Fragen ungeklärt. Dieses Arbeitspaket hat sich auf die Beschreibung der notwendigen Konzepte und Architektur für das Management von sicheren IP-basierten Kommunikationen konzentriert, es wurde jedoch kein Protokoll für den Ablauf des Frameworks wie in EVITA [30] entwickelt (z.B. für Policy Update, Rekeying-Prozess ...). Außerdem wurde keine Migrationsstrategie definiert, so dass weitere Überlegungen angestellt werden müssen, um die vorgeschlagene Security-Architektur mit tatsächlichen nicht-IP-basierten oder nicht-sicherheitsorientierten Systemen (wie z.B. AUTOSAR) zu verwenden oder sie entsprechend anzupassen. Endlich fehlt ein Best-Practice-Leitfaden für das Management dieses Frameworks. Viele Ansätze wurden in den jeweiligen Haupt-Security-Konzepten (Schlüssel, Policy, IDS, HSM) betrachtet; weitere Untersuchungen sind erforderlich, um für jeden Anwendungsfall ein entsprechendes Security-Management, das optimal die Framework-Fähigkeiten benützt, zu definieren.

BIBLIOGRAPHIE

[1] Apache ETCH: Project Overview. http://incubator.apache.org/etch/ .

[2] Badger, L., Sterne, D. F., Sherman, D. L., Walker, K. M., AND Haghighat, S. A. 1995a. Practical domain and type enforcement for UNIX. In Proceedings of the IEEE Symposium on Research in Security and Privacy (Oakland, CA, May). IEEE Computer Society Press, Los Alamitos, CA, 66–77.

[3] Bell, D., and LaPadula, L. Secure computer systems: A mathematical model. Tech. Rep. MTR-2547, Volume II, Mitre Corporation, Bedford, Massachusetts, 1973.

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

[5] Oliver Bubeck, Jens Gramm, Markus Ihle, Jamshid Shokrollahi, Robert Szerwinski and Martin Emele: „A Hardware Security Module for Engine Control Units", 9th escar Embedded Security in Cars Conference, Dresden, 2011.

[6] Corradi A., Montanari R., Tibaldi D., Toninelli A.: A Context-centric Security Middleware for Service Provisioning in Pervasive Computing. In *Proceedings of the The 2005 Symposium on Applications and the Internet* (SAINT '05). IEEE Computer Society, Washington, DC, USA, 421-429. DOI=10.1109/SAINT.2005.2.

[7] Cugola G., Jacobsen H.: Using publish/subscribe middleware for mobile systems. *SIGMOBILE Mob. Comput. Commun. Rev.* 6, 4 (October 2002), 25-33.

[8] Daza V., Domingo-Ferrer J., Sebé F. and Viejo A.: Trustworthy privacy preserving car-generated announcements in vehicular ad hoc networks. IEEE Transactions on Vehicular Technology, vol. 58, pp. 1876-1886, 2009.

[9] Dearle A., Kirby G., Norcross S., Macdonald A., Bigwood G.: Towards Adaptable and Adaptive Policy-Free Middleware. CoRR, vol 1006.3732, 2010.

[10] EMVY Design Specification (Fachkonzept) – BMW Group. September 2009.

[11] EVITA Project: http://evita-project.org/ .

[12] Graham, G. und Denning, P. 1972. Protection-principles and practice. In Proceedings on AFIPS Spring Joint Computer Conference. AFIPS Press, Arlington, VA, 417–429.

[13] Herstellerinitiative Software (HIS): „SHE – Secure Hardware Extension Version 1.1" – Functional Specification v1.1, rev439, 16.10.2009.

[14] Hofman S.(ed.) et al, "Sicherheit auf der Ebene der Kommunikationsinfrastruktur", SEIS Projekt, Ergebnisdokument AP4.2.

[15] Kohl J., Neuman C., The Kerberos Network Authentication Service, IETF-Standard, RFC1510, September 1993.

[16] Lamberty J. et al, „Sicherheitsanforderungen", SEIS Projekt, Ergebnisdokument AP1.3.

[17] Lamberty J. et al, „Sicherheitsmodell" , SEIS Projekt, Ergebnisdokument AP4.1.

[18] Leung A. and Mitchell C.: A service discovery threat model for ad hoc networks. in Proceedings of the International Conference on Security and Cryptography SECRYPT, 2006, Setubal, p7-10, INSTICC Press

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

[20] Manns D. et al, "Sicherheit bei der Kommunikation mit der Umwelt", SEIS Projekt, Ergebnisdokument AP4.4.

[21] Matthess M. et al, "Sicherheitsvalidierung", SEIS Projekt, Ergebnisdokument AP4.5.

[22] Moses T. (ed.): eXtensible Access Control Markup Language (XACML) Version 2.0. OASIS Standard, February 2005.

[23] Trabels S., Urvoy-Keller G., and Roudier Y.," On the Impact of DoS Attacks on Secure Service Discovery", embedded and ubiquitous computing , 532 – 537 ,international conference on 17-20,dec 2008.

[24] RSA Laboratories: "PKCS #11: Cryptographic Token Interface Standard", v2.20, 28. Juni 2004, at the time of writing publicly available at: ftp://ftp.rsasecurity.com/pub/pkcs/pkcs-11/v2-20/pkcs-11v2-20.pdf

[25] Trusted Computing Group, website, http://www.trustedcomputinggroup.org

[26] Trusted Computing Group (TCG), "Mobile Phone Work Group Mobile Trusted Module Specifivation", version 1.0, rev.7.02, 2010 at the time of writin publicly available at http://www.trustedcomputinggroup.org/resources/mobile_phone_work_group_mobile_trusted_module_specification

[27] Trusted Computing Group (TCG), "TPM Main Specification Version 1.2", (Part 1-3) ISO/IEC standard 11889 (1-4), at the time of writin publicly available at http://www.trustedcomputinggroup.org/resources/tpm_main_specification

[28] Thürmer D. et al, "CE-Gerät und Benutzerschnittstellen", SEIS Projekt, Ergebnisdokument AP3.2.

[29] Sandhu R. and Samarati P.: Access control: Principles and practice. IEEE Communications, 32(9):40-48, 1994.

[30] Schweppe H. et al., "Secure on-board protocols specification," EVITA Project, Tech. Rep. Deliverable D3.3, 2010.

[31] Weckemann K. et al, „Middelware", SEIS Projekt, Ergebnisdokument AP3.1.

[32] Weyl B. et al., "Secure on-board architecture specification," EVITA Project, Tech. Rep. Deliverable D3.2, 2010.