Masterarbeit Nr. 04190

Absicherung der SOME/IP Kommunikation bei Adaptive AUTOSAR

Jochen Kreissl

Studiengang:	Informatik
Prüfer/in:	Prof. Dr. Ralf Küsters
Betreuer/in:	Prof. Dr. Ralf Küsters
Beginn am:	22. Mai 2017
Beendet am:	15. November 2017
CR-Nummer:	• **Security and privacy~Distributed system security** • **Security and privacy~Security protocols** • *Networks~Network protocols* • *Computer system organization~Embedded systems*

Kurzfassung

Die Entwicklung einer neuen Generation vernetzter, (teil-)autonomer und zumindest teilweise elektrisch betriebener Fahrzeuge fordert von der Automobilindustrie den Wechsel zu einer neuen Fahrzeugarchitektur, welche den Einsatz dynamischer Softwarekomponenten auf leistungsstarker Hardware ermöglicht. Um den schnellen Austausch der notwendigen Informationen zwischen einzelnen Systemen zu gewährleisten, werden zudem On-Board Kommunikationsnetze mit hoher Bandbreite benötigt. Die *adaptive AUTomotive Open System ARchitecture* (AUTOSAR) Plattform in Verbindung mit IP-basierter, service-orientierter Kommunikation soll die Basis für diese neue Fahrzeuggeneration bereitstellen. Für eine hohe Abstraktionsebene der Kommunikation zwischen einzelnen Softwarekomponenten sieht die *adaptive AUTOSAR* Spezifikation den Einsatz der *Scalable service-Oriented MiddlewarE over IP* (SOME/IP) vor, welche den dynamischen Aufbau von Kommunikationskanälen zwischen den Komponenten zur Laufzeit des Systems ermöglicht (*Service Discovery*). Durch den hohen Grad der Vernetzung von Fahrzeugen, insbesondere durch die Internetanbindung via moderner Mobilfunkstandards, steigt zugleich die Gefahr von Angriffen auf Fahrzeuge durch Hacker und Schadsoftware. Um dennoch die Sicherheit der übertragenen Daten, und damit indirekt die Sicherheit der Passagiere, zu gewährleisten, müssen die eingesetzten Kommunikationsprotokolle höchsten Sicherheitsansprüchen genügen.
Nach einer kurzen Einführung der *adaptive AUTOSAR* Plattform und des SOME/IP-Protokolls wird in der folgenden Arbeit eine Gefahren- und Risikoanalyse der Fahrzeugarchitektur durchgeführt. Dabei liegt der Schwerpunkt auf der Analyse der On-Board Kommunikation. Weiterhin werden Sicherheitsprotokolle untersucht, welche die aufgedeckten Schwachstellen wirksam und effizient absichern, wobei auf den Einsatz asymmetrischer Verfahren soweit wie möglich verzichtet wird. Insbesondere werden Protokolle zur Absicherung von Multicast-basierter Kommunikation betrachtet, da das SOME/IP-Protokoll für die Implementierung effizienter Gruppenkommunikation und das Auffinden von Softwarekomponenten IP-Multicast einsetzt. Die betrachteten Protokolle werden im Anschluss auf ihre Kompatibilität mit dem SOME/IP-Standard untersucht und durch Kombination verschiedener Ansätze ein Gesamtkonzept für die Absicherung der gesamten SOME/IP-Kommunikation innerhalb des Systems entwickelt. Während die Unicast-Kommunikation mithilfe des weit verbreiteten *Transport Layer Security* (TLS) Protokoll erreicht werden kann, wird eine Kombination von TLS und dem *Time Efficient Stream Loss-tolerant Authentication* (TESLA) Protokoll vorgestellt, um die Multicast-Kommunikation von SOME/IP abzusichern. Wird die Option zur Sitzungswiederaufnahme (*Session Resumption*) des TLS-Protokolls genutzt, so kommt das vorgestellte Konzept vollkommen ohne asymmetrische Kryptographie aus und erreicht dennoch die Sicherheitseigenschaften Geheimhaltung und Senderauthentifizierung für alle Kommunikationskonzepte des SOME/IP Protokolls. Die Authentizität übertragener Nachrichten kann dabei insbesondere auch dann garantiert werden, wenn ein Angreifer vollständige Kontrolle über einen Kommunikationspartner besitzt.

Diese Arbeit wurde in Kooperation mit der **Vector Informatik GmbH** erstellt.
Die gewährten Einblicke in die Verfahren, Einschränkungen und Standards der Automobilindustrie, sowie dem Ist-Zustand moderner Fahrzeugsysteme, ermöglichten das Erstellen einer anwendungsnahen und zugleich theoretischen Arbeit.

Inhaltsverzeichnis

1 Einleitung

Mit Einführung von Funkschnittstellen (u.a. Bluetooth, WiFi, Mobilfunk) in aktuellen Automobilen und der gleichzeitig steigenden Komplexität der verbauten elektronischen Steuergeräte (engl. Electronic Control Unit, ECU) für die Bereiche Multimedia, Fahrassistenzsysteme und Motorsteuerung ergibt sich eine breite Angriffsfläche auf moderne Automobile. Ein potentieller Angreifer kann somit, ohne direkten physikalischen Zugriff auf das Fahrzeug, sicherheitskritische Systeme manipulieren und dadurch im schlimmsten Fall schwere Unfälle verursachen [1]. Die Auswirkungen eines solchen Angriffs werden durch die geplante, flächendeckende Einführung von Interfahrzeugkommunikation und Fahrzeug-Infrastruktur-Kommunikation weiter vergrößert. Zusammengefasst werden diese Technologien mit *Car-to-X* (C2X) abgekürzt. Da diese Verfahren als zentraler Baustein für die Realisierung autonomer Fahrzeuge betrachtet werden, erhalten Angriffsmöglichkeiten auf diese Kommunikationspfade eine weitreichende sicherheitskritische Bedeutung.
Berichte über entdeckte Schwachstellen in ECUs durch Sicherheitsforscher, Interessierte und selbst Laien zeigen deutlich, dass die Möglichkeit eines solchen Angriffs bereits für die aktuelle Fahrzeuggeneration sehr real ist. Ein entscheidender Parameter für die Auswirkungen eines erfolgreichen Angriffs, ist die Zeitspanne, welche zur Beseitigung entdeckter Sicherheitslücken seitens der Hersteller benötigt wird. Erste Verfahren für automatische, drahtlose Updates (*Flash-over-the-Air*, OTA) werden von Herstellern - insbesondere im Premiumsektor - teilweise eingesetzt, doch im Allgemeinen müssen Fahrzeuge im Falle eines Softwarefehlers vom Fahrzeughalter in eine (Vertrags-)Werkstatt gebracht werden, um dort eine aktualisierte Softwareversion zu erhalten. Im Fall einer Zero-Day Sicherheitslücke kann ein Angreifer von einer aktiven Zeitspanne von zumindest mehreren Wochen ausgehen, bis Gegenmaßnahmen den Großteil der betroffenen Fahrzeuge erreicht haben. Dies ist im Gegensatz zu der Reaktionszeit auf kritische Sicherheitslücken in modernen Softwareprodukten zu sehen, für welche in der Regel bereits nach wenigen Stunden, spätestens aber nach wenigen Tagen, ein entsprechender Patch bereitsteht.

Um der Komplexität eines modernen Automobils gerecht zu werden, setzen Hersteller vermehrt auf standardisierte Plattformen wie *AUTOSAR*, welche das Gesamtsystem stark von der verwendeten Hardware abstrahieren und Komponenten modularisieren. Mit *adaptive AUTOSAR* befindet sich aktuell eine neue Plattform in der Standardisierung, welche den Anforderungen der zukünftigen Automobilgeneration gerecht werden soll. Um die benötigte Bandbreite für den internen Datenaustausch sicherzustellen, wird Ethernet als zentrale Kommunikationstechnologie verwendet werden. Mit dem IP-basierten SOME/IP (engl. scalable service-oriented middleware over IP) wird ein Kommunikationsprotokoll eingesetzt, welches eine schnelle und

dynamische Kommunikation der einzelnen Komponenten gewährleisten soll.
Während der SOME/IP Standard die Funktionalität des Protokolls vollständig abdeckt, enthält dieser keinerlei Absicherung gegen Beeinflussungen durch einen Angreifer. Ziel dieser Arbeit ist es deshalb, die Sicherheitseigenschaften des Systems zu analysieren und geeignete Sicherheitsmechanismen zur Absicherung der fahrzeuginternen, SOME/IP-basierten Kommunikation zu untersuchen.

1.1 Historische Entwicklung

Für das Verständnis des betrachteten Systems, sowie dessen einzigartige Anforderungen und Einschränkungen, hilft eine kurze Betrachtung der historischen Entwicklung der in Automobilen verbauten Technik. Im Folgenden wird diese Entwicklung kurz aufgezeigt.

1.1.1 Erster Einsatz von elektronischen Steuergeräten

In den 1970er Jahren begannen erste Automobilhersteller einzelne ECUs in ihre Fahrzeuge zu verbauen [2] [3]. Wenige Kilobyte Programmcode steuerten stark spezialisierte, strikt getrennte Funktionen - zu Beginn vor allem die Treibstoffeinspritzung oder allgemein die Ansteuerung des Motors. Die durch ECUs erreichte Präzision ermöglichte eine erhebliche Steigerung der Effizienz von Verbrennungsmotoren und so die geringen Verbrauchs-, Lärmentwicklungs- und Schadstoffwerte moderner Fahrzeuggenerationen [3]. Weitere Steuergeräte kamen in den 1980er Jahren hinzu, um komplexe Sicherheitsvorrichtungen (z.B. ABS, ESP und Airbags) zur Vermeidung von schweren Unfällen in Fahrzeuge zu integrieren. Neben Funktionen zur Steigerung der Effizienz und Betriebssicherheit, kamen zusätzlich auch Komfort- und Multimediasysteme zum Einsatz (z.B. Sitzheizung, elektronische Fensterheber, Radio und CD-Player). Insbesondere diese letzten Teilbereiche bieten Herstellern heute ein weites Feld zur Alleinstellung ihrer verschiedenen Modelle gegenüber denen der Konkurrenz.
Diese Entwicklung führt dazu, dass über 80 % aller Neuerungen in der Automobilbranche auf Softwaresysteme zurückzuführen sind [3]. Der hohe Wettbewerbsdruck kombiniert mit legislativen Anforderungen zur Adaption moderner Techniken zur Steigerung der Betriebssicherheit und zur Einschränkung der Emissionen (sowohl mit Hinblick auf Lärmentwicklung als auch Schadstoffausstoß), ließ die Anzahl der ECUs rasch ansteigen. Jede einzelne dieser ECUs besaß eigene, über nicht-standardisierte Direktverbindungen angeschlossene Sensoren und Aktuatoren, um die entsprechenden Funktion zu verwirklichen. Oft wurden Sensoren von mehreren ECUs verwendet, oder einzelne Steuergeräte über Punkt-zu-Punkt Verbindungen verbunden, um komplexere Funktionen zu implementieren und die benötigten Daten auszutauschen.

1.1.2 Standardisierte On-Board Kommunikation

Um die steigende Zahl nicht-standardisierter Kabelverbindungen zu reduzieren (zur Gewicht- und Kosteneinsparung) und zugleich eine einfache Kommunikation zwischen verschiedenen ECUs zum Austausch benötigter Informationen zu ermöglichen, entwickelte und standardisierte die Firma Bosch im Jahr 1986 den *Controller Area Network* (CAN) Bus [4]. Das Bussystem zusammen mit dem zugehörigen Kommunikationsprotokoll ist in den ISO-Normen 11898-2 bzw. 11898-3 standardisiert und avancierte in der Folge rasch zu dem verbreitetsten Feldbus in der Automobilindustrie. Da diese bis heute eine beherrschende Rolle in der Kommunikationsinfrastruktur von Fahrzeugen einnimmt - insbesondere für sicherheitskritische Steuergeräte - soll im Folgenden ein kurzer Überblick über dessen Eigenschaften und Funktionsweise gegeben werden.
Die zentralen Merkmale des CAN-Bus sind die Linientopologie der Netzwerkknoten und die Übertragung aller Daten nach dem Broadcast-Prinzip mit anschließender Selektion durch die Empfänger. Dies bedeutet, dass jeder Netzwerkknoten eines CAN-Bus grundsätzlich alle gesendeten Daten empfängt und daraus diejenigen auswählt, welche für die jeweilige Anwendung von Interesse sind. Direkter Vorteil dieses Prinzips ist die implizite Teilung aller Ressourcen und Informationen innerhalb des Netzes ohne Mehrfachübertragungen.
Die empfängerseitige Selektion erfolgt dabei anhand des 11-Bit langen *Object Identifier*, der sowohl die Priorität als auch den Inhalt einer CAN-Nachricht definiert. Nachrichten höherer Priorität verdrängen dabei solche niedrigerer Priorität direkt auf dem Bus (sog. bitweise Arbitrierung), wodurch ein expliziter Busmaster oder Controller entfällt. Eine explizite Kennzeichnung von Sender oder Empfänger ist im ursprünglichen CAN-Protokoll nicht vorgesehen, kann jedoch von spezialisierten Protokollen durch geeignete Nutzung des *Object Identifier* erreicht werden. Dafür wird insbesondere die *Extended Frame Format* Erweiterung verwendet, welche die Verwendung eines 29-Bit Identifiers erlaubt. Ein Beispiel für ein Protokoll stellt der SAE J1939 Standard zur Übermittlung von Diagnosedaten via CAN dar, welche die zusätzlichen Bits des Extended Frame zur Kodierung von Absender und Empfänger verwendet. [5].
Zur Sicherstellung der korrekten Übertragung nutzt das CAN-Protokoll eine 16-Bit CRC (engl. cyclic redundancy check) und ein Bestätigungsverfahren durch die Empfänger. Erkennt ein Empfänger einen Fehler, setzt dieser das Fehlerbit am Ende der Transmission, welches alle anderen Empfänger veranlasst den fehlerhaften Frame zu verwerfen, um Datenkonsistenz im Netzwerk zu sichern.
Im wesentlichen existieren zwei CAN-Bus Varianten, welche sich lediglich in der Übertragungsgeschwindigkeit unterscheiden. Der sogenannte Komfortbus wird zumeist zur Steuerung von Komfortfunktionen eingesetzt, wie zum Beispiel Fensterheber oder Sitzkontrolle, und besitzt eine Übertragungsrate von 125 KBit pro Sekunde. Die Powertrain genannte Variante kommt dagegen zumeist zur Steuerung von sicherheitskritischen Funktionen des Fahrzeugs (z. B. Motor-, Brems-, Airbag- und Fahrwerksteuerung) zum Einsatz und besitzt eine Datenrate von 1 MBit pro Sekunde. Beide Varianten können pro Frame maximal 8 Byte Nutzdaten übertragen. In den meisten Fahrzeugen kommen aufgrund der beschränkten Datenrate mehrere CAN-Bus Netze zum Einsatz. Ist ein Datenaustausch zwischen verschiedenen Netzen notwendig, so kann

die mithilfe von Bridges oder implizit durch *Multi-Connected* ECUs erreicht werden. Letzteres beschreibt eine ECU welche aufgrund ihrer Funktion an mehrere CAN-Busse angeschlossen ist und somit als indirekte Bridge wirken kann. Klassisches Beispiel für eine solche ECU stellen die Head-Units dar, welche dem Fahrer alle wichtigen Fahrzeuginformationen zur Verfügung stellen. Neben den kritischen Informationen, etwa zur Drehzahl des Motors oder Warnleuchten bei diversen Fehlern, welche typischerweise über einen oder mehrere Powertrain-Netze übertragen werden, zeigen Head-Units auch Informationen aus diversen Komfortbus-Netzwerken an. Dies kann zum Beispiel eine Anzeige der Außentemperatur oder Gurtkontrolle sein.

Eine Erweiterung des CAN-Bus durch Bosch im Jahr 2011 (Standardisiert als ISO-11898-1 im Jahr 2012) erlaubt eine flexible Datenrate (FD). Das Protokoll bleibt bis auf die maximale Länge der Nutzdaten und Übertragungsrate komplett identisch und ist mit klassischen CAN-Netzwerken vollkommen interoperabel. Die maximale Länge der Nutzdaten wurde bei CAN-FD von 8 auf 64 Bytes erhöht.

1.1.3 Diagnose und Test

Um die Einhaltung von Abgasgrenzwerten während des Betriebs des Fahrzeugs zu kontrollieren und zu protokollieren ist in den USA seit 1994 gesetzlich festgelegt, dass Neuzulassungen den *On-Board Diagnose* (OBD-II) Standard unterstützen müssen. Abgasrelevante Fehler (z. B. überdurchschnittlicher Verbrauch, Fehlzündungen, hohe Leerlaufwerte) werden dazu von den verantwortlichen ECUs in Log-Dateien unter einer standardisierten Fehlerkennung gespeichert und können durch das OBD-II Protokoll von einem Tester ausgelesen werden. Der standardisierte OBD-II Port erlaubt unter anderem auch den Zugriff auf den CAN-Bus des Fahrzeugs, um Diagnosebefehle und notwendige Informationen auszulesen, die über den Bus gesendet werden. In der EU wurde mit dem europäischen Pendant EOBD im Jahr 2001 eine vergleichbare legislative Regelung eingeführt. Der *Unified Diagnostics Services* (UDS) ISO-Standard stellt eine Erweiterung dieses Prinzips dar und ist in der Industrie weit verbreitet. UDS beschreibt - neben den OBD-II Befehlen und Fehlermeldungen - eine abstrahierte Client-Server-Kommunikation. Dabei ist der Umfang der Kommunikation vom Hersteller der einzelnen ECUs frei wählbar. Weit verbreitete (Diagnose-)Services sind *get* und *set*-Methoden für Datenfelder oder allgemeine Speicherbereiche, Wechsel des Betriebsmodus (z. B. in den Diagnose-, oder Debugmodus) und Updatefunktionalität. Diese weitreichenden Möglichkeiten ermöglichen eine bessere Test- und Wartbarkeit der Systeme. Insbesondere die Fähigkeit zu *in-system*-Updates ist entscheidend, da andernfalls Steuergeräte erst aus und nach dem Updateprozess wieder in das System eingebaut werden müssten. In der Regel erfolgt die UDS-Kommunikation direkt über den CAN-Bus des Fahrzeugs.

1.2 Grundlegende Systemarchitektur

Die grundlegende Architektur - eine Vielzahl von ECUs mit dedizierten Aufgaben, sowie Datenaustausch über ein oder mehrere Feldbusse und Bridges - findet bis heute Anwendung. Die Zahl der verbauten ECUs variiert dabei stark, wobei in Luxusmodellen zumeist deutlich mehr Steuergeräte verbaut sind als in Einsteigermodellen. Doch selbst in günstigen Modellen moderner Baureihen sind 70 und mehr ECUs verbaut, in Spitzenmodelle dagegen oftmals (weit) über 100 verschiedene Steuergeräte[6]. Diese hohe Zahl an ECUs und der damit einhergehenden Menge an Software sorgt für ein hochkomplexes, verteiltes System. Zusätzlich zu der Anzahl der ECUs, steigt auch die Komplexität der verwendeten Software. Lag diese zu Beginn der Entwicklung noch bei wenigen hundert Zeilen Programmcode für das ganze Fahrzeug, steigerte sich dies bereits im Jahr 1981 auf etwa 50.000 und liegt bei modernen Autos bei bis zu einhundert Millionen Zeilen Code und hunderten Megabyte an Maschinencode (vgl. Abbildung 1.1).
Allein die enorme Größe der verwendeten Software spricht für die Existenz einer ganzen Reihe von Fehlern, welche potentiell von einem Angreifer ausgenutzt werden können.

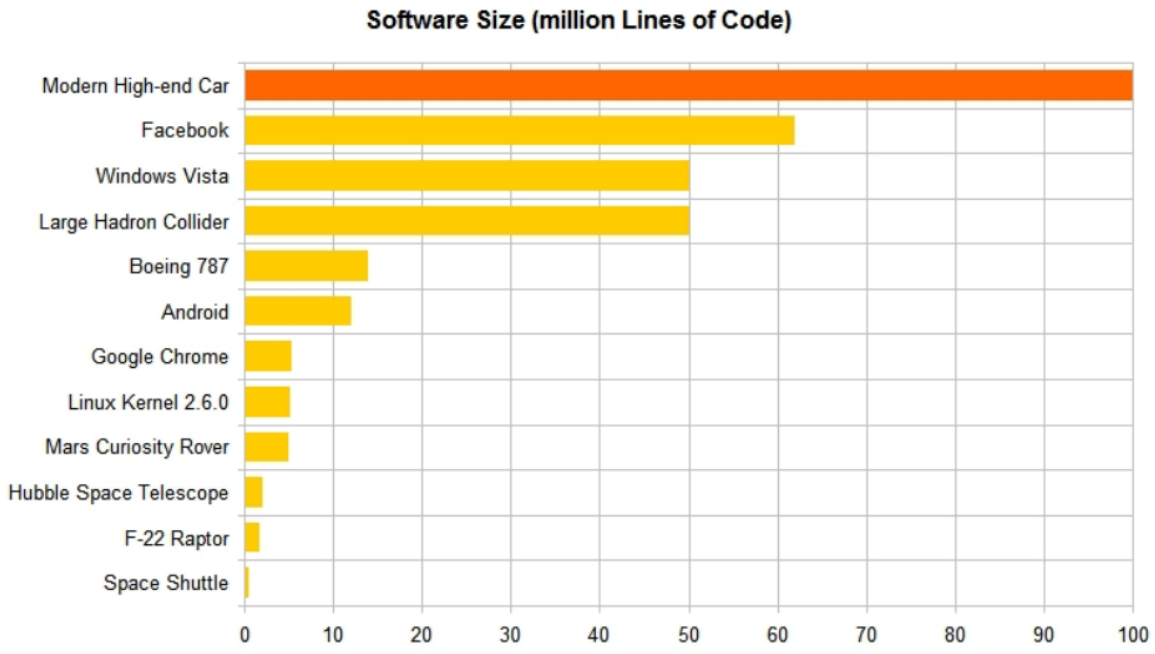


Abbildung 1.1: Vergleich der Größe verschiedener Softwareprodukte und der Menge aller Software eines typischen Automobils [7] (Stand: 2009).

1.3 Die *AUTOSAR* Plattform

Zur Beherrschung der enormen Komplexität, Verbesserung der Zusammenarbeit von Zulieferern und Automobilherstellern sowie Kosteneinsparung durch einfache Wiederverwendbarkeit bildeten eine Reihe namhafter Fahrzeughersteller und Zulieferer (u.a. Daimler, BMW, Bosch, Volkswagen, Toyota, Ford, GM) im Jahr 2003 die AUTomotive Open System ARchitecture (AUTOSAR) Initiative [8][9]. Ziel der Initiative ist eine standardisierte, modularisierte und skalierbare Systemarchitektur für Fahrzeuge zu entwickeln. Zentraler Fokus ist die Abstraktion der funktionellen Softwarekomponenten von der verbauten Hardware und die Spezifikation der dazu notwendigen Schnittstellen. Dies ermöglicht eine einfachere Wiederverwertbarkeit und Austauschbarkeit von Software und Hardware zwischen Ausstattungsvarianten, Modellreihen, Herstellern und Zulieferern. So ist es beispielsweise möglich denselben Code für die Steuerung der elektrischen Fensterheber für alle Fahrzeugtypen mit vier Scheiben zu verwenden, obwohl die zugrundeliegende Fahrzeugarchitektur völlig verschieden ist. Eine solche AUTOSAR Applikation wird als *Software Component* (SWC) bezeichnet und bildet die eigentliche Funktion der jeweiligen Steuergeräte dar.
Die *AUTOSAR classic* Architektur (vgl. Abb. 1.2) beschreibt drei klar getrennte Software-Ebenen, wobei höhere Schichten - ähnlich dem OSI-Schichtenmodell - tiefere Ebenen über ein standardisiertes Interface nutzen und so zu einer steigenden Abstraktion und Modularisierung beitragen.

1. Die oberste Ebene wird als Application Layer bezeichnet, da diese die Software enthält, welche die eigentlichen Funktionen bereitstellt. Über das AUTOSAR Interface erhalten Applikationen dabei Zugriff auf Funktionen der Laufzeitumgebung und auf die abstrahierten Fähigkeiten der jeweiligen Hardware. Dadurch lassen sich AUTOSAR Applikation unabhängig von der verbauten Hardware entwickeln.

2. Als Middleware befindet sich die sog. Runtime Environment (RTE) in der zweiten Schicht. Diese stellt eine abstrakte Kommunikation zwischen Applikationen bereit - unabhängig ob die tatsächliche Kommunikation über ein reales Netzwerk oder lediglich on-chip (z.B. via Memory-Mapped I/O) abläuft. Das dabei verwendete Konzept eines virtuellen Busses wird als Virtual Function Bus (VFB) bezeichnet.

3. Software der untersten Ebene wird als Basissoftwarekomponente (engl. Basic Software Component, BSC) bezeichnet. Abstrakt gesehen fällt darunter alle Software, welche die abstrakte Schnittstelle für höheren Schichten bereitstellt und diese auf entsprechende Funktionen der vorliegenden Hardware implementiert. Dabei zerfallen BSCs in zwei weitere Gruppen: AUTOSAR Standard Software und ECU spezifische Software. Standardisierte Software ist z. B. ein kompatibles Betriebssystem, ein Kommunikationsframework (Sammlung von Interfaces für die typischen Netzwerkkomponenten), Speicherabstraktion und Systemservices (z.B. Diagnose, Event-Logging). ECU spezifische Software beschreibt Module, welche einen Zugriff auf spezielle Hardwarefunktionen

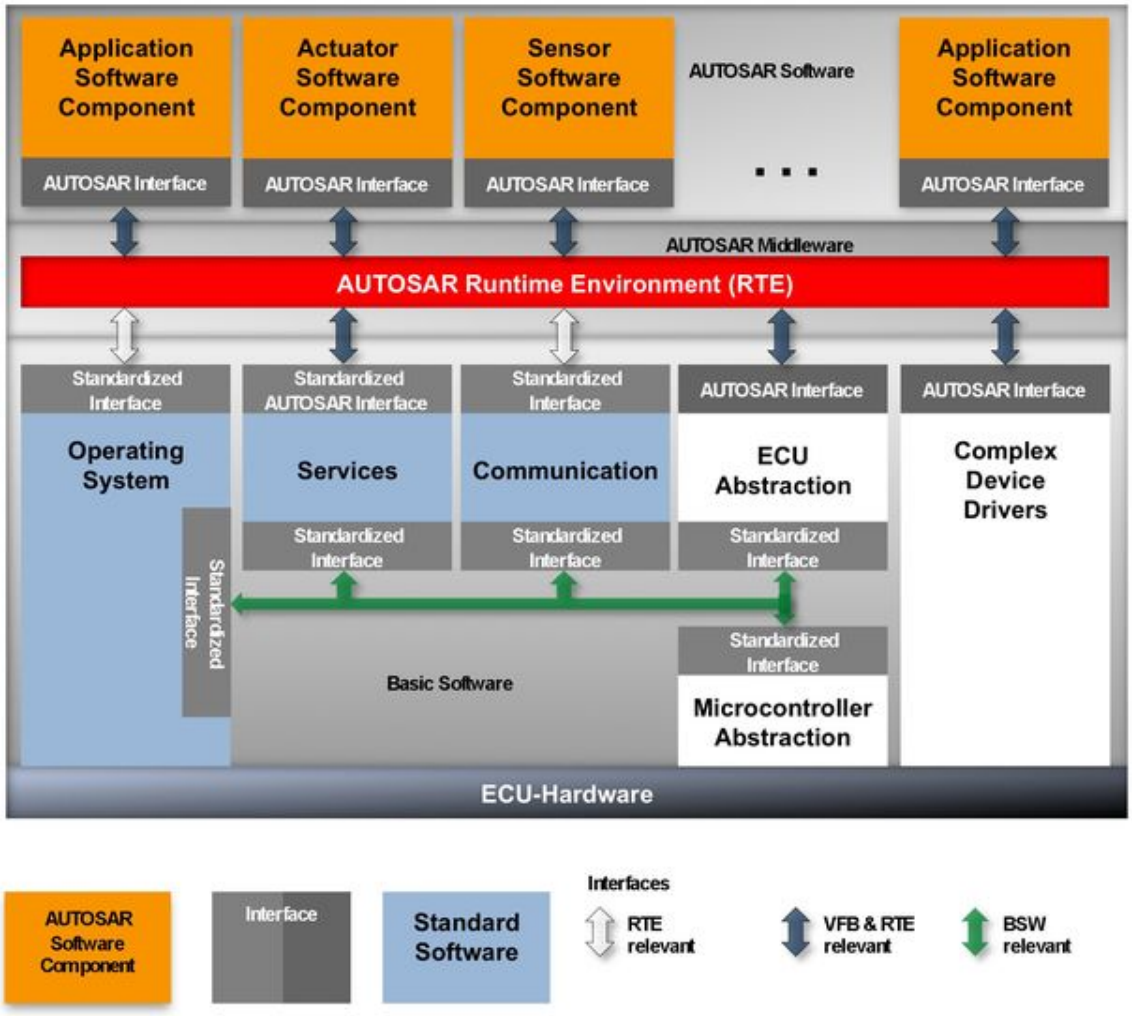


Abbildung 1.2: Systemarchitektur der AUTOSAR Classic Plattform[†]. Softwarekomponenten werden in drei Schichten unterteilt: Application Layer, Runtime Environment und Basic Software Components.

für höhere Schichten bereitstellen. Dies kann etwa der Treiber für eine nicht standardisierte Funkschnittstelle oder die direkte Nutzung von Befehlssatzerweiterungen zur Performanzsteigerung spezieller Operationen sein.

Der AUTOSAR Standard beschreibt zudem eine komplexe Methodik für Spezifikation, Generierung und Konfiguration des Gesamtsystems. Aus dieser (Gesamt-)Systembeschreibung werden

[†]http://www.autosar.org/fileadmin/_processed_/csm_AUTOSAR_ECU_softw_arch_b_fbf24b2245.jpg

Extrakte für die einzelnen ECUs entnommen - entsprechend dem jeweiligen Anteil einer ECU am Gesamtsystem. Die Applikationsentwicklung ist dabei bewusst von dieser Methodik losgelöst. Lediglich die Schnittstellen der Applikation fließen als *SWC-Description* in den AUTOSAR Prozess ein und definiert die Schnittstellen, welche die SWC benötigt - ohne den internen Aufbau der Komponente genauer zu beschreiben. Dadurch ist eine einfache Integration von SWCs unterschiedlicher Hersteller möglich, ohne dass deren geistiges Eigentum (engl. Intellectual Property, IP) - und damit wirtschaftliche Interessen - verletzt werden.

2 Grundlagen

Im folgenden Kapitel werden die für die Arbeit notwendigen Grundlagen eingeführt. Der erste Abschnitt beschriebt das *adaptive AUTOSAR*, welches das zugrundeliegende System für die Arbeit darstellt. Im Anschluss werden eine Reihe von Angriffsszenarien vorgestellt, welche für aktuelle Fahrzeuge bekannt sind. Daraus lassen sich typische Angriffsvektoren ableiten, welche für die Definition des Angreifermodells und die Durchführung der Sicherheitsanalyse von Bedeutung sind. Im Abschnitt 2.3.1 werden die notwendigen kryptographischen Begriffe und Primitiven und deren Notation definiert, welche im weiteren Verlauf der Arbeit eingesetzt werden.

2.1 Zukünftige Fahrzeugarchitektur

Aktuelle Trends in der Automobilbranche sorgen für erhöhte Anforderungen bezüglich Rechenleistung, Schnittstellenvielfalt, Bandbreite der fahrzeuginternen Kommunikation und Dynamik der Softwarearchitektur. Zu den treibenden Trends [10] zählen die Folgenden:

E-Mobilität Elektrische Fahrzeuge werden, trotz aller Herausforderungen, als Zukunft des Automobils betrachtet [10]. Neben all den Anforderungen an Fahrzeuge mit herkömmlichen Verbrennungsmotoren ist die Energieeffizienz für elektrische Fahrzeuge von herausragender Bedeutung, um die Reichweite nicht (noch) weiter zu verkürzen. Eine der Möglichkeiten zur Reduktion des Stromverbrauchs ist nicht benötigte Funktionen und Steuergeräte in einen stromsparenden Stand-By Modus zu versetzen und durch Zentralisierung von Funktionen die Gesamtzahl der Steuergeräte zu reduzieren.

(Teil-)Autonomes Fahren Je autonomer ein Fahrzeug funktionieren soll, desto höher die Anforderungen an eine schnelle Verarbeitung von umfangreichen Sensordaten (z. B. Radar und Kamerabilder) und den Austausch dieser Daten zwischen verschiedenen Teilsystemen. Dies erfordert erhebliche Rechenleistung der verbauten Hardware und Bandbreite der fahrzeuginternen Kommunikation. Bereits in aktuellen Fahrzeugen werden immer komplexere Fahrerassistenzsysteme (engl. advanced driver assistence systems, ADAS) eingesetzt. Zu diesen zählen unter anderen Notbremsassistent, Fahrbahnerkennung und Spurhalteassistenz, Verkehrszeichenerkennung, Einparkhilfen und Einparkautomatik.

Vernetztes Fahrzeug In der modernen, vernetzten Welt ist die weitreichende Vernetzung der Fahrzeuge ein naheliegender Schritt, um in dem starken Wettbewerbsgefüge neue

Akzente und Anreize für die Kunden zu setzen. Dabei lassen sich drei unterschiedliche Vernetzungstypen unterscheiden:

- *Car-2-X* (auch: V2X) bezeichnet die Vernetzung von Fahrzeugen untereinander und mit spezieller Infrastruktur (z. B. Ampelanlagen, Verkehrsleitzentralen, Straßenschilder). Dieser Vernetzung verspricht, im Falle einer flächendeckenden Einführung, einen großen Beitrag zur Vermeidung von Verkehrsunfällen durch das frühzeitige Erkennen von Gefahren und Hindernissen zu leisten. So kann etwa bei einer Notbremsung eine Warnmeldung an alle nahen Fahrzeuge gesendet werden, um die nachfolgenden Fahrer auf diese Gefahr aufmerksam zu machen, oder ein Warnhinweis auf ein nahendes Einsatzfahrzeug eingeblendet werden. Zudem bietet die Vernetzung bessere Möglichkeiten zur Verkehrsplanung und -überwachung, etwa durch frühzeitiges Entdecken von Staus oder dem Aufspüren von kritischen Engstellen im Straßennetz. Auch für den Aufbau eines elektronischen Mautsystems ist eine Kommunikation zwischen Fahrzeugen und Infrastruktur erforderlich. Für *Car-2-X* werden seit Jahren internationale Standards für die Kommunikationsprotokolle und eingesetzte Funktechnik erstellt [11]. Erste Serienmodelle werden bereits mit *Car-2-X* Fähigkeiten ausgeliefert [12].
  Für *Car-2-X* muss das Fahrzeug in der Lage sein, empfangene Daten zweifelsfrei zu verifizieren, damit keine Entscheidungen auf Grundlage veralteter, beschädigter oder gefälschter Informationen getroffen werden. Dies bedingt den Einsatz kryptographischer Operationen zur Verifikation der Nachrichten, welche innerhalb einer harten Zeitgrenze durchgeführt werden muss. Dies erfordert erhebliche Rechenleistung, um diese Anforderung auch bei hohem Nachrichtenaufkommen (z. B. einem Stau oder innerhalb einer Großstadt) einhalten zu können.

- Die flächendeckende Verbreitung von Funknetzen erlaubt es den jeweiligen Herstellern Daten von Fahrzeugen zu sammeln und in einer Cloud auszuwerten oder dem Nutzer bereitzustellen. Dies ermöglicht, neben Multimediaanwendungen wie z. B. Aktualisierung von Kartenmaterial oder Internetanbindung, auch die Kontrolle spezieller Diagnosewerte oder das Entdecken von Fehlern und anschließende Verteilung notwendiger Updates, ohne dass der Halter des Fahrzeugs dafür eine Werkstatt aufsuchen muss. Insbesondere dieser letzte Punkt bietet einen erheblichen Vorteil, um unvermeidliche Softwarefehler schnell und auf allen betroffenen Fahrzeugen zu beheben.
  Ein weiterer Anwendungsfall ist die Bereitstellung von Fernzugriff auf bestimmte Funktionen für Benutzer, Hersteller und autorisierte Drittparteien (z. B. Ortung und Aktivierung der Wegfahrsperre im Fall eines Diebstahls, Analyse von Fahrdaten für Versicherungen). Einige Versicherungen bieten Preismodelle an, welche die Versicherungsgebühren anhand des aufgezeichneten und ausgewerteten Fahrstils des Fahrers anpassen [13].
  Systeme für autorisierten Fernzugriff werden bereits heute von vielen Herstellern (insb. für Fahrzeuge aus der Premiumklasse) angeboten, z. B. OnDrive von GM [14] oder ConnectedDrive von BMW [15].

- Dynamische Softwarekomponenten und Vernetzung mit dem Internet der Dinge (IoT). Bereits heute besitzen die meisten modernen Fahrzeuge verschiedene Schnittstellen zur Verbindung von Endgeräten der (Mit-)Fahrer, z. B. Bluetooth zur Verbindung von Smartphones, oder eine USB-Schnittstelle.
  Analog zur Entwicklung im Mobiltelefonie-Markt erwarten Hersteller in Zukunft mehr Umsatz mit dem sogenannten digitalen Ökosystem Automobil zu erwirtschaften, als mit dem bloßen Verkauf von Fahrzeugen [10]. Denkbare Beispiele für solche dynamischen Komponenten sind etwa kurzzeitig aktive Programme zur Steigerung der Fahrzeugleistung (z. B. für einen Wochenendausflug) oder gesetzlich vorgeschriebene Software für bestimmte Fahrzeugtypen (z. B. zur Emissionskontrolle während der Fahrt, oder Mautsysteme).
  Für dieses Ziel muss eine zukünftige Fahrzeugplattform hochdynamische Softwarekomponenten unterstützten. Dies erfordert offene Standards und Schnittstellen, um die Entwicklung kompatibler Software zu vereinfachen und somit das Gesamtsystem attraktiv zu machen.
  Ein zusätzlicher, treibender Aspekt ist die Integration von Fahrzeugen in ein zukünftiges IoT-Gesamtnetz. In Verbindung mit dem Aspekt der Elektromobilität ergeben sich direkte Überschneidungen, z. B. die Integration in ein intelligentes Strommanagement. So könnte etwa der Ladevorgang des Fahrzeugs über die Nachtstunden durch den Strombetreiber soweit verzögert werden, bis das Netz ausreichend Kapazitäten zur Verfügung hat - dies hilft Stromspitzen zu vermeiden und die Netzstabilität zu wahren. Denkbar sind auch Ansätze, die Speicherkapazitäten von elektrisch betriebenen Fahrzeugen als Stromspeicher für das Stromnetz einsetzen [16].

2.1.1 Die *adaptive AUTOSAR* Plattform

Die Motivation für eine neue Systemarchitektur (auf Basis des bewährten AUTOSAR Prinzips) ist direkt aus den neuen Anforderungen abzuleiten [17]. Hinzu kommt das Ziel der Steuergerätekonsolidierung, um die hohe Zahl der verbauten ECUs zu verringern, indem die lange gültige Semantik der dedizierten Steuergeräte zugunsten einer zentralisierten Architektur aufgegeben wird.
Insbesondere die Anforderung an eine hochdynamische Softwarearchitektur, welche sich aus den vorgestellten Trends ergeben, ist mit der *AUTOSAR classic* Architektur nicht möglich. Dies ist jedoch erforderlich, um dynamisch neue Applikationen einzufügen (oder zu entfernen), oder die Allokation der Applikationen auf die ausführende ECU zur Laufzeit zu ändern.
Die neue *adaptive AUTOSAR* Plattform basiert für diesen Zweck auf serviceorientierter (auch: diensteorientierter) Kommunikation, um eine noch stärkere Unabhängigkeit von der zugrundeliegenden Hardware und anderer Softwarekomponenten zu erreichen [18]. In *AUTOSAR classic* dagegen werden *ECU-Extract* und *SWC Description* lokal innerhalb einer ROM-Komponente auf der betreffenden ECU statisch gespeichert. Diese definieren zusammen das Verhalten der ECU innerhalb des Gesamtsystems und können nicht trivial ausgetauscht werden.
In der neuen Plattform besitzt dagegen jede Applikation eine sog. *Manifest*-Datei, die alle

Schnittstellen, Methoden und angebotene sowie benötigte Services der Applikation beschreibt. Zusammen mit dem Maschinencode der Applikation bildet diese eine Einheit, welche auf einer beliebigen *adaptive AUTOSAR*-ECU ausgeführt werden kann. Sowohl Manifest als auch Applikationscode werden persistent gespeichert und zur Ausführung in den RAM der ausführenden ECU geladen. Dabei wird jede Applikation in einem getrennten virtuellen Adressbereich ausgeführt, um Wechselwirkungen zwischen Applikationen auszuschließen.

Die zugrundeliegende Plattform ist ein (zumindest PSE51-konformes) POSIX-Betriebssystem, wodurch kompatible Anwendungen beliebig auf vorhandene ECUs ausgeführt werden können. Applikationen werden als getrennte Prozesse ausgeführt und können mittels standardisierter Bibliotheken und Schnittstellen auf Funktionen des Betriebssystem und der abstrahierten Hardware zugreifen [19]. Abbildung 2.1 zeigt den logischen Aufbau der *adaptive AUTOSAR* Plattform. Die Kommunikation mit anderen Applikationen (Prozessen) erfolgt lokalitätstrans-

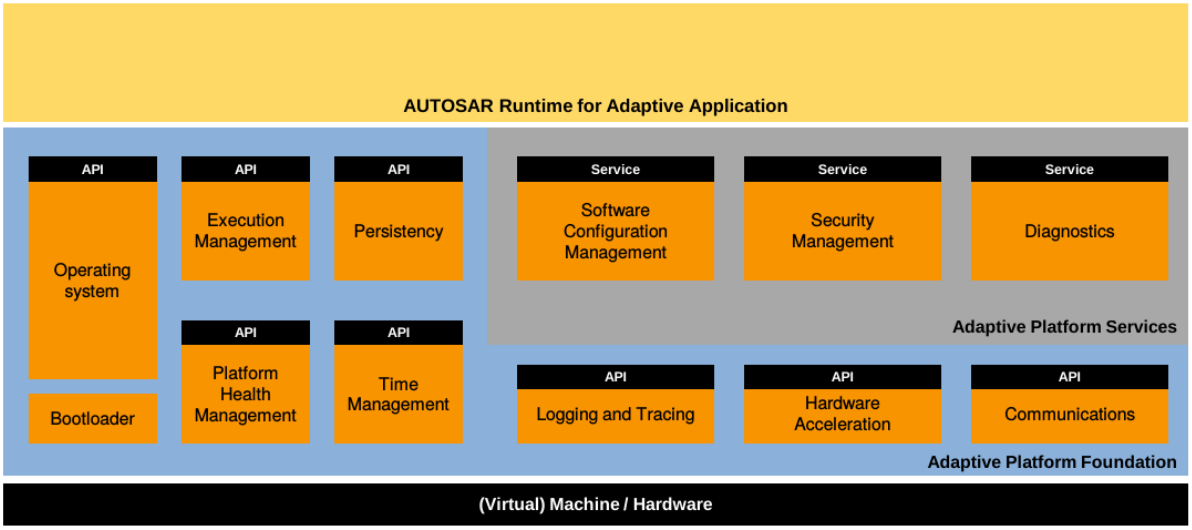


Abbildung 2.1: Die Adaptive AUTOSAR Plattform.

parent, d.h. Kommunikation mit Applikationen auf entfernten ECUs erfolgt identisch (aus Sicht der Anwendung), wie eine Verbindung zu einer Applikation, welche auf derselben ECU ausgeführt wird. Kommunikationswege zwischen Applikationen können somit zur Laufzeit ermittelt werden - dies ermöglicht eine dynamische (Re-)Allokation von Applikationen, ohne die Funktionalität anderer Applikationen negativ zu beeinflussen. Zur Realisierung dieser Funktionalität wird das SOME/IP Protokoll als Middleware eingesetzt. Das Protokoll wird in Kapitel 3 vorgestellt und im weiteren Verlauf der Arbeit untersucht.

2.1.2 Beispielarchitektur

Abbildung 2.2 zeigt eine mögliche Fahrzeugarchitektur, bei der sowohl klassische als auch *adaptive AUTOSAR* Steuergeräte Verwendung finden. Eine solche (oder ähnliche) Architektur ist für die erste Generation von Serienfahrzeugen mit *adaptive AUTOSAR* vorstellbar. Die Beibehaltung der CAN-Bus gesteuerten *AUTOSAR classic*-ECUs für den *Powertrain*-Cluster (u.a. Motorsteuerung, Bremssysteme, Airbags) ist auf die Echtzeitanforderung und absolute Robustheit dieser sicherheitsrelevanten Komponenten zurückzuführen, welche auf der *adaptive* Plattform nur schwer zu erfüllen sind. Zudem erfordern diese Komponenten eine vergleichsweise geringe Datenrate, um ihre Funktionen zu erfüllen - im Gegensatz zu komplexen ADAS-Systemen (z. B. Spurhalteassistent oder Einparkassistent) und Sensorsystemen (z. B. Radar oder Kameras). Die Verbindung zwischen den Plattformen wird durch ein zentrales

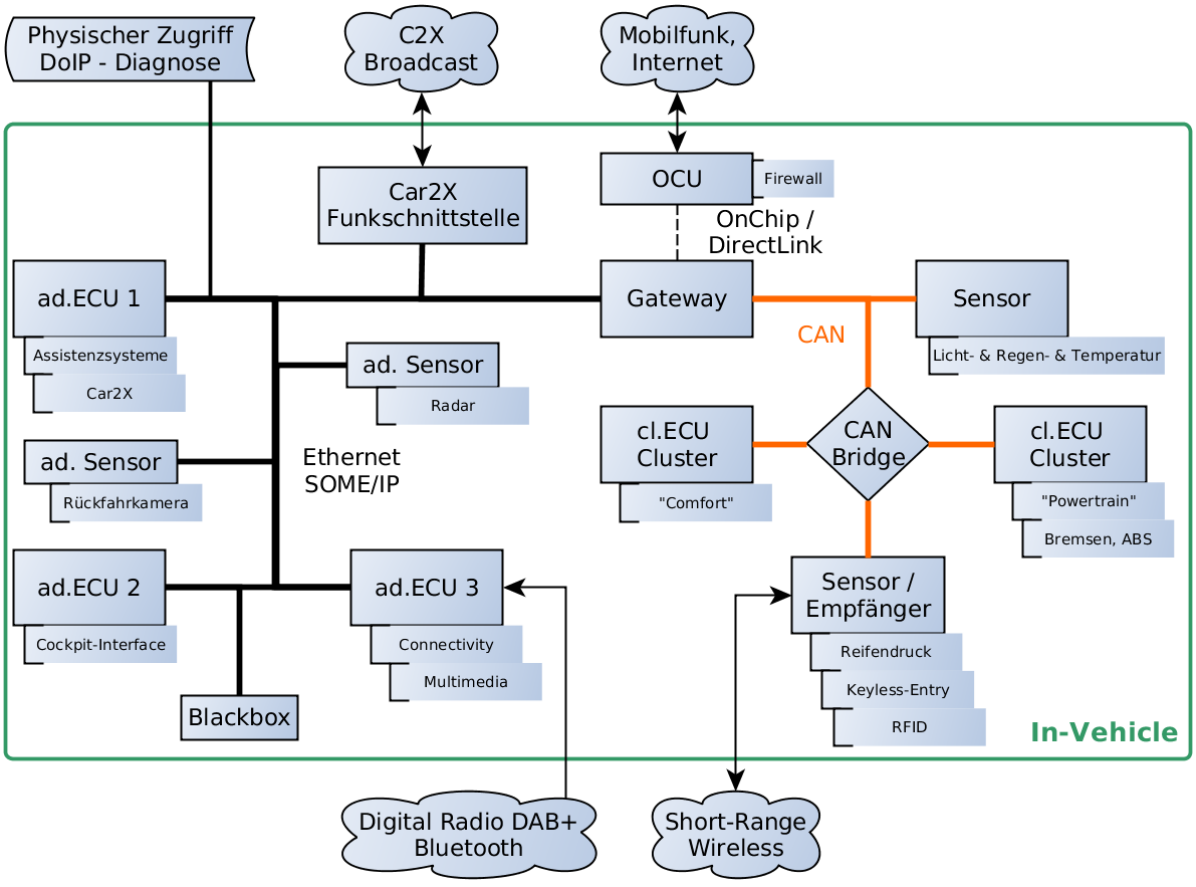


Abbildung 2.2: Mögliche Systemarchitektur bei Verwendung von klassischen und *adaptiven* ECUs.

Gateway (auch Head-Unit) hergestellt. Nachrichten können so zwischen der signalbasierten Kommunikation der CAN-Bus basierten *AUTOSAR classic* Architektur und der serviceori-

entierten Kommunikation der ethernetbasierten *adaptive AUTOSAR* Architektur übersetzt werden.

Für Beispiele und Erklärungen wird im Folgenden ein vereinfachtes Modell verwendet. Das Modell in Abbildung 2.3 besteht aus den fünf Kommunikationsknotenpunkten $\Pi_{Gateway}, \Pi_{ADAS}, \Pi_{Connectivity}, \Pi_{HMI}$ und Π_{Sensor}. Jeder Knoten repräsentiert dabei eine physisch getrennte ECU und sämtliche Kommunikation zwischen den Knoten erfolgt mittels SOME/IP. Kapitel 3.1.1 beschreibt eine Reihe von *adaptive AUTOSAR* Applikationen, welche auf diesen Knoten verteilt sind. Für das System wichtige Schnittstellen zu anderen Onboard-Netzwerken bzw. Funkschnittstellen sind als abstrakte Wolken eingezeichnet.

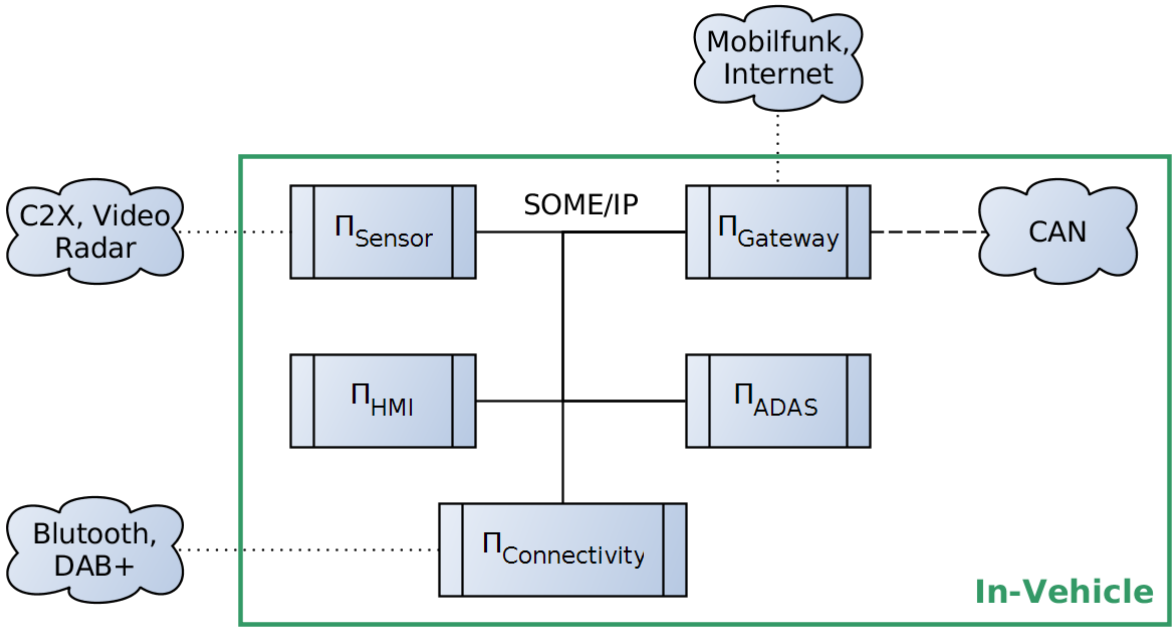


Abbildung 2.3: Vereinfachtes Graphmodell. Durchgezogene Verbindungen bezeichnen das Ethernet-basierte SOME/IP Netzwerk. Gepunktete Linien weisen auf eine Funkschnittstelle hin, während die gestrichelte Linie eine kabelgebundene Anbindung (mittels CAN-Bus) bezeichnet.

2.1.2.1 Beispielhardware

Eine mögliche Wahl für eine *adaptive AUTOSAR* Hardwareplattform stellt das *MinnowBoard* dar, welche von der MinnowBoard Foundation angeboten wird [20][21][22]. Die Non-Profit-Organisation bietet die Intel-x86 basierten SoC-Boards als Open-Source Hardware an, inklusive aller zugehöriger Software und langfristigem Firmware-Support. Erklärtes Anwendungsfeld der Minnow-Board-Serie ist das IoT-Umfeld, Industrieautomatisierung und Automobil bzw. generell Transportsysteme.

Das System bietet in der Grundversion einen x86 64-Bit Dual-Core Intel-Atom Prozessor mit einer maximalen Taktung von $1,46$GHz und 2GB DDR3-RAM und einer vielzahl industrietypischer I/O-Schnittstellen, unter anderem zweimal Gigabit-Ethernet. Eine Quad-Core Variante mit bis zu $1,91$GHz steht ebenfalls zur Verfügung. Die eingesetzte E3826-CPU besitzt, neben den genannten Kenngrößen und einer sehr niedrigen Leistungsaufnahme von durchschnittlich 7 Watt, zudem eine Reihe nützlicher Befehlssatzerweiterungen, welche für den Anwendungsfall von Vorteil sind. Insbesondere sind hier die Intel Virtualisierungstechnik VT-x und der AES-Befehlssatz AES-NI zu nennen, da sowohl Virtualisierung als auch effiziente Verschlüsselung für ein sicheres und performantes System von herausragender Bedeutung sind.
Aufgrund der x86-64-Bit Architektur ist eine Kompatibilität mit den meisten Betriebssystemen und einer Vielzahl von Softwareprodukten ohne zusätzliche Anpassungen möglich. So ist beispielsweise der Einsatz einer speziellen POSIX-Variante, angepasst an die Anforderungen des Systems, auf Basis eines MinnowBoards denkbar.

2.2 Angriffsziel: Fahrzeuge

Die Steuerung nahezu sämtlicher Funktionen des Fahrzeugs durch elektronische Steuergeräte und somit Software impliziert die Möglichkeit von gezielter Beeinflussung durch einen Angreifer mithilfe informationstechnischer Mittel. Eine Reihe von Arbeiten [1][23][24][25][26][27][28][29] zeigt, in welchem erheblichen Umfang eine solche Manipulation möglich ist.
Im Wesentlichen unterscheiden sich die Angriffe dabei nach dem Zugriffstyp. Die folgenden Kapitel beschreiben verschiedene Zugriffstypen, welche einem angenommenen Angreifer zur Verfügung stehen.

2.2.1 Physischer Zugriff

Der fehlende Schutz gegen Beeinflussung von ECUs und Onboard-Kommunikation ermöglicht einem Angreifer mit physischem Zugriff, praktisch beliebige Aktionen auszulösen [23][26][28][29]. Für moderne Fahrzeuge ist physischer Zugriff praktisch gleichbedeutend mit Zugriff auf den CAN-Bus. Dieser kann z. B. mittels standardisierter Diagnosegeräte erfolgen (vgl. Abschnitt 1.1.3).
Drei Hauptargumente erklären diese offensichtliche Schwachstelle.

- Eingebettete Systeme unterliegen klassischerweise erheblichen Ressourceneinschränkungen, was die Anwendung komplexer kryptographischer Verfahren unwirtschaftlich macht (vgl. Abschnitt 2.3.4). Zudem ist der CAN-Bus aufgrund dessen niedrigen Übertragungsrate und Beschaffenheit kaum kryptographisch abzusichern.

- Legislative Anforderung für offene Diagnose und Wartung. Hersteller von Fahrzeugen (und ECUs) müssen gesetzliche Vorschriften beachten, wonach neben den Vertragswerkstätten der Hersteller auch freie Werkstätten in der Lage sein müssen Diagnose, Reparaturen und Updates durchführen zu können [23]. Folglich muss der Zugang zu Diagnosetools, benötigten Systeminformationen und eventuell genutzten kryptographischen Schlüsseln den freien Werkstätten zur Verfügung gestellt werden. Diese Anforderung stellt eine fundamentale Schwierigkeit für sichere und autorisierte Diagnosezugriffe sowie die generelle Systemsicherheit dar.
- Kosten-Nutzen-Abwägung. Sämtliche ECUs und Kommunikationsbusse sind innerhalb des Fahrzeugs verbaut und damit von der Karosserie (und Schließmechanismen) vor Zugriff von außen geschützt. Dies bedingt einen längeren, invasiven und damit leicht feststellbaren Zugriff auf das Fahrzeug.
  Ein Angreifer mit länger währendem physischem Zugriff auf das Fahrzeug und den notwendigen Mitteln für invasive Maßnahmen könnte auch einen rein mechanischen Angriff durchführen (ersetzen einer ECU, physische Manipulation von Systemen z. B. der Bremsleitung). Ein solcher Angriff ist praktisch nicht zu verhindern und erfordert zugleich ein geringeres Fachwissen des Angreifers, da keine Kenntnisse über die IT-Systeme des Fahrzeugs benötigt werden. Aus einer Kosten-Nutzen-Abwägung ist daher eine Absicherung gegen physische Angriff auf IT-Systeme nicht wirtschaftlich sinnvoll, da ein Angriff mit vergleichbarer Schwere und denselben Voraussetzungen existiert, welcher praktisch nicht verhindert werden kann.

2.2.2 Indirekter physischer Zugriff

Checkoway et al. beschreiben eine Reihe von indirekten physischen Angriffen auf Fahrzeuge, mithilfe von manipulierten CDs oder USB-Datenträger, welche dem Halter zugespielt werden [1]. Werden die Datenträger an das Fahrzeug angeschlossen kann eine manipulierte Firmware auf das Fahrzeug aufgespielt werden, welche dem Angreifer Kontrolle über das System verschafft.
Ein weiterer Ansatz beschreibt die Manipulation eines marktüblichen Diagnosegeräts mittels dessen WiFi-Schnittstelle. Die Autoren demonstrierten die möglichen Konsequenzen eines solchen Angriffs, indem sie einen funktionsfähigen Wurm programmierten, welcher in der Lage war von einem Diagnosegerät aus alle verwundbaren Fahrzeuge und Diagnosegeräte zu befallen - und auch von betroffenen Fahrzeugen auf unbefallene Diagnosegeräte zu gelangen. Zusammen mit einer vorkonfigurierten Umweltbedingung (Datum und Mindestgeschwindigkeit) könnte sich ein solcher Wurm über einen langen Zeitraum praktisch unentdeckt verbreiten und dann zu einem bestimmten Zeitpunkt eine vom Angreifer gewünschte Funktion ausführen. Die Autoren demonstrierten zudem, dass das Schadprogramm sich im Anschluss selbst löschen kann und nach einem erzwungenen Neustart der befallenen ECUs praktisch nicht mehr nachweisbar ist.

2.2.3 Drahtloskommunikation

Mit dem Einsatz von drahtlosen Kommunikationsschnittstellen in Fahrzeugen entfällt jedoch die in Abschnitt 2.2.1 beschriebene Annahme des notwendigen physischen Zugriffs. Gelingt es dem Angreifer eine solche Funkschnittstelle zu kompromittieren, kann er einen Angriff auf das Fahrzeug führen, ohne direkten physischen Zugriff auf das System zu besitzen [1].
Funkschnittstellen lassen sich dabei entsprechend in zwei Kategorien einteilen:

Kurze Reichweite Zu dieser Kategorie zählen Bluetooth, WiFi, RFID und NFC Schnittstellen. Bluetooth ist der de Facto Standard für die Verbindung von Mobiltelefonen mit Fahrzeugen für Multimediazwecke und Freisprecheinrichtungen und besitzt typischerweise eine Reichweite von 10 Meter. Aufgrund der Komplexität des Protokolls (und einer Reihe von Versionen) existieren selbst in weit verbreiteten Implementierungen immer noch Sicherheitslücken. Eine solche Lücke kann einem Angreifer Vollzugriff auf die entsprechende ECU (oftmals die Head-Unit) und darüber vollständige Systemkontrolle verschaffen.
WiFi wird aktuell zumeist für das Anbieten eines Internetzugangs für die Insassen genutzt. In zukünftigen Fahrzeuggenerationen wird zudem eine WiFi-Variante (IEEE 802.11p) das Medium für die C2X-Kommunikation darstellen (vgl. 2.1). Neben der Möglichkeit, auf diesem Weg Kontrolle über das System zu erlangen, sind für die C2X-Anwendung auch andere Angriffe denkbar - z. B. der sogenannte *virtuelle Steinewerfer* (in Anlehnung an Personen, welche Steine auf vorbeifahrende Fahrzeuge werfen). Durch das Senden gefälschter Nachrichten könnte nahen Fahrzeugen eine Gefahr vorgetäuscht werden, die nicht existiert. Je nach Inhalt der Nachricht könnte ein Assistenzsystem daraufhin eine gefährdende Reaktion (z. B. eine Notbremsung bei hoher Geschwindigkeit) einleiten, um der vorgetäuschten Gefahr auszuweichen. Die Reichweite beträgt zwischen 300 und 1500m (je Hop, Multi-Hop möglich) [30].
RFID und NFC Funkschnittstellen werden für Funkschlüssel und *Keyless Go*-Systeme eingesetzt. Zudem können sie für die gesetzlich vorgeschriebene Reifendruckkontrolle eingesetzt werden. Eine Sicherheitsuntersuchung eines solchen Systems durch Roufa et al. zeigte erhebliche Sicherheitslücken auf [25]. Neben erheblichen Implikationen für den Datenschutz (Fahrzeug-Tracking ist mit einfachen Mitteln innerhalb von etwa 40m möglich) gelang es den Autoren zudem die zugeordnete ECU mittels gefälschter Nachrichten zur Anzeige falscher Informationen und zum Neustart zu zwingen. Dies legt das Vorhandensein einer erheblichen Sicherheitslücke nahe, welche es einem Angreifer unter Umständen erlauben könnte vollständige Systemkontrolle zu erlangen.

Weite Reichweite Zu diesen Schnittstellen zählen überregionale Kommunikationstechniken, insbesondere Mobilfunknetze, das GPS sowie analoge und digitale Radioübertragung. In dieser Kategorie sind aufgrund der weiten Verbreitung und Attraktivität für Angreifer insbesondere der digitale Radiostandard DAB+ und Mobilfunk von herausragender Bedeutung. Grund dafür ist die einfache Verfügbarkeit von *Software-Defined Radio* Komponenten, welche das Emulieren von (Mobil-)Funkstationen mithilfe von Software

ermöglicht. Der Vorzug von Mobilfunk gegenüber DAB+ ist lediglich die Möglichkeit zur bidirektionalen Datenübertragung, welche bei DAB+ nicht gegeben ist.
Miller und Valasek beschreiben einen als Jeep Hack bekannt gewordenen Angriff auf einen Jeep Cherokee über das Mobilfunknetz - ohne regionale Einschränkung [24][31]. Den Autoren gelang es eine Schwachstelle in der Head-Unit auszunutzen, um eine an diese angeschlossene CAN-Bridge mit einer manipulierten Firmware zu versehen. Nach einem Neustart der betroffenen ECU waren die Autoren in der Lage beliebige CAN-Frames in das Bordnetz des Fahrzeugs über die Mobilfunkverbindung der Head-Unit einzuspielen. Die dadurch ausgelösten Fehlfunktionen reichten von der Aktivierung der Scheibenwaschanlage bis zur Kontrolle über (einzelne) Bremsen und vollständiger Lenkkontrolle bei niedrigen Geschwindigkeiten und Rückwärtsfahrt. Die folgende Rückrufaktion betraf 1,4 Millionen Fahrzeuge verschiedener Typen und Baujahre. In einer früherer Veröffentlichung der Autoren wird ein ähnlicher Angriff auf einen anderen Fahrzeugtyp beschrieben [1].
Auch über DAB+ sind ähnliche Angriffe möglich [32]. Insbesondere die verwendete Software zur Darstellung von Bildern, welche über DAB+ empfangen werden, zeigte sich als Schwachstelle. Diese konnte vom Autor zur Erlangung der vollständige Kontrolle über die Empfangseinheit ausgenutzt werden. Im Anschluss wäre ein Angriff auf das Gesamtsystem analog zu dem von Miller und Valasek beschriebenen möglich.

2.3 Informations- und Betriebssicherheit

Ein großer Teil der ECUs eines Fahrzeugs sind direkt (z.B. ABS) oder indirekt (z.B. Airbag-Sensoren) sicherheitskritisch und eine Störung der erwarteten Funktion kann zu schweren Unfällen führen. Selbst Systeme, welche auf den ersten Blick nicht sicherheitskritisch sind, z.B. Multimediasysteme oder Sitzsteuerung, können im Fall einer starken und unerwarteten Fehlfunktion, z.B. durch extreme Lautstärke, den Fahrer erschrecken oder stören und somit einen Unfall verursachen.
Gründe für das Fehlverhalten einer Komponente können in zwei Kategorien zusammengefasst werden: Hardware- oder Softwarefehler und gezielte Beeinflussung. Um Fehler in Hardware und Software zu vermeiden, werden eingesetzte Komponenten, sowohl während des Herstellungsprozesses als auch in regelmäßigen Abständen innerhalb der Lebenszeit (z.B. während des Startvorgangs), komplexen Tests unterzogen. Bei derart komplexen, verteilten Systemen, wie moderne Fahrzeuge sie darstellen, lassen sich Fehler dennoch praktisch nicht ausschließen. Im Folgenden wird nicht auf Vermeidung dieser Fehlerquelle eingegangen, da der Fokus der Arbeit auf Maßnahmen gegen gezielte Beeinflussung liegt. Tatsächlich bedingen jedoch oftmals Softwarefehler die Anfälligkeit eines Systems für gezielte Beeinflussung.

2.3.1 Informationssicherheit

Die Abwehr gezielter Angriffe auf ein Informationstechnisches (IT) System wird als Informationssicherheit bezeichnet. Informationssicherheit stellt ein Oberbegriff für die folgenden Sicherheitseigenschaften eines IT-Systems.

1. Vertraulichkeit (engl. confidetiality) bezeichnet die Eigenschaft, dass die Daten eines Systems nicht von unautorisierten Personen gelesen oder verbreitet werden können. Dies gilt insbesondere auch für die Datenübertragung.
2. Integrität (engl. integrity) fordert die Nachvollziehbarkeit von Änderungen der Daten, insbesondere dürfen keine unbemerkte Veränderungen vorgenommen werden.
3. Verfügbarkeit (engl. availability) ist die Garantie, dass das System bereit und nicht ausgefallen ist. Diese Eigenschaft wird meist mit einer Wahrscheinlichkeit angegeben, da eine 100-prozentige Garantie im Allgemeinen nicht möglich ist.

Diese Sicherheitseigenschaften werden als allgemeine oder grundlegende Eigenschaften bezeichnet, da ohne sie ein sicheres System nicht möglich ist.
Je nach Anwendungsgebiet können zudem eine Reihe weiterer Sicherheitseigenschaften definiert werden.

4. Authentizität (engl. authenticity) bezeichnet die Überprüfbarkeit der Herkunft bzw. Vertrauenswürdigkeit von Daten und stellt einen Spezialfall der Datenintegrität dar. Authentifizierte Kommunikation garantiert einem Empfänger, dass empfangene Daten von dem erwarteten Absender gesendet und nicht von einer dritten Partei manipuliert wurden.
5. Autorisierung (engl. authorization) beschreibt die Prüfung der Berechtigung zur Durchführung einer Aktion.
6. Verbindlichkeit (engl. nonrepudiation) bezeichnet die Eigenschaft, dass eine ausgeführte Handlung von der ausführenden Partei nicht plausibel geleugnet werden kann. Eine Handlung kann folglich beweisbar einer bestimmten Partei zugeordnet werden. Diese Eigenschaft ist insbesondere für rechtliche Zwecke von Interesse.
7. Datenschutz (engl. privacy) bezeichnet Maßnahmen zum (rechtlich vorgeschriebenen) Schutz personenbezogener Daten vor Missbrauch.

2.3.2 Kryptographie

Zur Sicherstellung der Sicherheitseigenschaften eines IT-Systems werden Methoden der Kryptographie eingesetzt. Im Folgenden werden kurz die kryptographischen Primitiven vorgestellt, welche zur Sicherstellung der Informationssicherheit eingesetzt werden.

2.3.2.1 Verschlüsselungsverfahren

Verschlüsselungstechniken ermöglichen es einen Klartext P (engl. Plaintext) mittels einer geheimen Information K (engl. Key) in einen chiffrierten Text C (engl. Cipher) zu überführen und umgekehrt [33]. Für kryptographisch sichere Verfahren ist es ohne Kenntnis der geheimen Information K nicht ohne erheblichen Aufwand möglich, aus C die ursprüngliche Information P abzuleiten. Verschlüsselungstechniken können folglich zur Wahrung des Sicherheitskriteriums Vertraulichkeit (und des eng verwandten Datenschutzes) eingesetzt werden, da die Daten - nach einer Verschlüsselung - nicht im Klartext vorliegen.
Spezielle Verfahren erfüllen überdies noch weitere Kriterien, z. B. kann das AES (engl. Advanced Encryption Standart) Verschlüsselungsverfahren im GCM (engl. Galoy Counter Mode) Betriebsmodus zusätzlich die Integrität der verschlüsselten Daten sicherstellen. Ein solcher Betriebsmodus wird mit AEAD (engl. *Authenticated Encryption with Associated Data*) oder AE (engl. *Authenticated Encryption*) bezeichnet. Zumeist sehen diese Verfahren die Möglichkeit vor, bestimmte Bereiche einer Nachricht lediglich in die Berechnung der Authentifizierung miteinzubeziehen, nicht jedoch zu verschlüsseln. Dies ist insbesondere für Headerinformationen hilfreich, welche zumeist für eine korrekte Übertragung der Nachricht im Klartext vorliegen müssen. Da die Authentizität von Headerinformationen und verschlüsselten Nutzdaten kombiniert wird, kann somit sichergestellt werden, dass eine Nachricht nur akzeptiert wird, falls auch die zugehörigen Headerinformationen unverändert vorliegen.

Verschlüsselungsverfahren werden in symmetrische und asymmetrische Verfahren unterteilt:

Symmetrisch Verschlüsselungstechniken dieser Kategorie benutzen denselben symmetrischen Schlüssel K für das Ver- und Entschlüsseln des Klartextes bzw. des Chiffretextes. Von zentraler Bedeutung für die Sicherheit symmetrischer Verfahren ist der verwendete Schlüssel, da ein Angreifer, der im Besitz des Schlüssels ist, die Verschlüsselung problemlos umkehren kann. Geheime Schlüssel können z.B. mittels einer Schlüsselderivationsfunktion (engl. Key Derivation Funktion, KDF) aus einem Langzeitschlüssel oder einer anderen geheimen Quelle gewonnen werden. Durch die Verwendung einer KDF zur Schlüsselerzeugung lässt sich die Lebenszeit eines Langzeitschlüssels erheblich verlängern, da dieser nicht direkt für kryptographische Operationen eingesetzt wird, sondern nur die von ihm abgeleiteten Schlüssel. Ein Großteil der symmetrischen Verfahren basiert auf sog. Blockchiffren, so auch das verbreitete AES Verfahren. Dabei wird die Eingabe in gleich große Blöcke aufgeteilt, welche nacheinander verschiedenen mathematischen Operationen und Transpositionen (vertauschen bestimmter Positionen) unterzogen werden.
Die folgenden Notation wird für die Anwendung einer symmetrischen Verschlüsselung (gekennzeichnet durch das s-Superscript) unter Verwendung des Schlüssels k auf den Klartext P eingesetzt: $C = \{P\}_k^s$.

Asymmetrisch Kennzeichen asymmetrischer Verfahren (engl. *Public Key Cryptography*) ist der Einsatz zweier unterschiedlicher Schlüssel für das Ver- und Entschlüsseln. Asymme-

trische Verfahren basieren auf mathematischen Problemen ohne rechnerisch effiziente Lösungsverfahren, wie z.B. die Primfaktorzerlegung sehr großer Zahlen im Fall des RSA-Verfahrens oder dem Problem des Diskreten Logarithmus. Einer der Schlüssel ist privat (auch: geheim, engl. Secret, k) und wird zum Entschlüsseln genutzt, der zweite ist öffentlich (auch: bekannt, engl. Known, k^{-1}) und wird nur zum Verschlüsseln eingesetzt. Nur der private Schlüssel muss geheim gehalten werden, der öffentliche kann weitergegeben werden. Ein Schlüsselpaar (k, k^{-1}) wird durch eine Schlüsselerzeugungsfunktion (engl. Key Generation Function, KGF) aus einer zufällig gewählten Zahl berechnet. Das entscheidende Sicherheitskriterium von Public-Key Verfahren ist, dass es praktisch unmöglich ist aus dem öffentlichen Schlüssel den geheimen Schlüssel zu berechnen.
Im Unterschied zu symmetrischen Verfahren genügt bei der Übertragung des öffentlichen Schlüssels die Wahrung der Authentifizierung - insbesondere bedeutet dies, dass der Schlüssel im Klartext übertragen und gespeichert werden kann. Der private Schlüssel muss jedoch analog zu symmetrischen Schlüsseln geheim gehalten und sicher gespeichert werden. Der Nachteil asymmetrischer Verfahren liegt in der höheren rechnerischen Komplexität der Algorithmen (Faktor 1000 und höher im Vergleich zu AES [34]) und des deutlich niedrigeren Sicherheitsniveaus für Schlüssel derselben Länge, was die Verwendung von erheblich größeren Schlüsseln zur Folge hat (vgl. Tabelle 2.1). Die Größe der Schlüssel und die erhebliche rechnerische Komplexität der Verfahren macht die Anwendung auf leistungsschwachen eingebetteten Systemen problematisch.
Aus diesem Grund werden asymmetrische Verfahren in der Praxis vor allem für den Aufbau einer sicheren Verbindung und den Austausch eines symmetrischen Sitzungsschlüssels verwendet und auf leistungsschwachen Systemen nach Möglichkeit vermieden.
Für die Anwendung eines asymmetrischen Verfahrens (gekennzeichnet durch das a-Superscript), wird die folgende Notation verwendet: $C = \{P\}^a_{k^{-1}}$ für Verschlüsselung mit dem öffentlichen Schlüssel bzw. $P = \{C\}^a_k$ für das Entschlüsseln mit dem privaten Schlüssel.

2.3.2.2 Kryptographische Schlüssel

Die Sicherheit kryptographischer Verfahren hängt maßgeblich von der Qualität des gewählten Schlüsselmaterials und dem sicheren Umgang damit ab. Ein Sicherheitsmaßstab für kryptographische Schlüssel stellt das sogenannte Sicherheitsniveau dar, welches von der Schlüssellänge und dem verwendeten Verfahren abhängig ist und in Bits angegeben wird. Genauer beschreibt das Sicherheitsniveau die Größe des Suchraums für den besten, zur Zeit bekannten Angriff auf das gegebene Verfahren. Liegt diese unter einer gewissen Schranke (welche sich aus der theoretisch möglichen Rechenleistung eines Angreifers bestimmt), wird das Verfahren als nicht (mehr) sicher eingestuft. Für die meisten Verfahren gilt die einfache Tatsache, dass eine Erhöhung der Schlüssellänge das erreichte Sicherheitsniveau steigert, wobei die Steigerung im Verhältnis zur Verlängerung des Schlüssels nicht proportional sein muss. Deutlich wird dies bei einem Vergleich des Sicherheitsniveaus verschiedener empfohlenen Verfahren (vgl. Tabelle 2.1) [35][36].

Sicherheitsniveau	Symmetrisch	Diffie-Hellman	RSA	ECC & ECDH
≤ 80 Bit	2DES (112)	1024	1024	160-223
112	3DES (168)	2048	2048	224-255
128	AES-128	3072	3072	256-383
192	AES-192	7680	7680	384-511
256	AES-256	15360	15360	512+

Tabelle 2.1: Vergleich der benötigten Schlüssellänge verschiedener kryptographischer Verfahren zum Erreichen eines gegebenen Sicherheitsniveaus. DES-Betriebsarten sind nur als Vergleichswert angegeben, die Verwendung wird durch das BSI nicht empfohlen.

Für symmetrische Verfahren (z. B. AES) wird *aktuell* eine Schlüssellänge von mindestens 128 Bit empfohlen, für asymmetrische Verfahren (z. B. RSA) eine Mindestschlüssellänge von 2048 Bit, für längerfristige Einsatzzwecke 3072 Bit [35]. Asymmetrische Kryptographie auf Basis elliptischer Kurven (ECC), besitzen ein erheblich höheres Sicherheitsniveau bei gleicher Schlüssellänge im Vergleich zu RSA-basierten Verfahren. Gegenüber symmetrischen Verfahren benötigt ECC etwa die doppelte Schlüssellänge zum Erreichen desselben Sicherheitsniveaus. Für Systeme mit langer Lebenszeit (über das Jahr 2023 hinaus) sind Prognosen für die notwendigen Sicherheitsniveaus schwer zu treffen, da die Entwicklung von zukünftigen Angriffen und der zur Verfügung stehenden Rechenleistung nur mittelfristig abzuschätzen sind [35]. Zuständige Behörden (NIST/BSI) empfehlen für solche Systeme bereits bei der Entwicklung die notwendigen Vorkehrungen zu treffen, um während der Laufzeit die verwendeten Schlüssellängen und Algorithmen auswechseln zu können. Da die typische Lebensdauer von Nutzfahrzeugen im Allgemeinen mindestens zehn Jahre beträgt, fallen Automobile (und damit das betrachtete System) offensichtlich in diese Kategorie.

Neben der reinen Schlüssellänge ist auch auf die Zusammensetzung der Schlüssel zu achten, dieser sollte nicht trivial zu erraten sein (z.B. Geburtstag des Besitzers, Name des Herstellers, Baujahr des Fahrzeugs, 0-Vektor o.ä.).
Für den sicheren Umgang mit Schlüsselmaterial gibt es klare Voraussetzungen, die ein Sicherheitssystem unterstützen sollte:

Verbreitung Geheime Schlüssel sollten so wenigen Parteien wie möglich bekannt sein, um das Risiko eines Bekanntwerdens zu minimieren. Eine geringe Verbreitung der Schlüssel bedeutet zudem, dass es für Angreifer weniger wirtschaftlich ist kostenintensive physikalische Angriffe einzusetzen, um Informationen über den verwendeten Schlüssel zu erhalten, da nur wenige Geräte den Schlüssel nutzen. Insbesondere bedeutet dies, das Geräteschlüssel nicht mehrfach innerhalb einer Fahrzeugreihe, oder gar über alle Fahrzeuge eines bestimmten Herstellers hinweg, zur Verwendung kommen (sog. Flottenschlüssel).

Schlüsselwechsel Eine ganze Reihe von Argumenten spricht für eine begrenzte Lebensdauer von Schlüsselmaterial - im Wesentlichen mit dem Ziel die Zeitspanne, welche einem

möglichen Angreifer zur Verfügung steht, zu begrenzen [36]. Verschiedene Kryptoanalyseverfahren benötigen eine erhebliche Anzahl an beobachteten kryptographischen Operationen oder eine gewisse Beobachtungszeitspanne, um ein gegebenes Verfahren zu brechen. Durch einen (rechtzeitigen) Schlüsselwechsel kann solchen Angriffen entgegen gewirkt werden. Eine eingeschränkte Nutzungsdauer schränkt zudem die Auswirkungen eines öffentlich gewordenen Schlüssels ein, da nur Daten dieser Zeitspanne von der Offenlegung betroffen sind, nicht jedoch Daten aus anderen Intervallen: je kürzer die Nutzungsdauer, desto geringer die Auswirkungen. Die Nutzungsdauer (engl. Cryptoperiod) eines Schlüssels hängt dabei von vielen Faktoren ab und kann zwischen wenigen Minuten und mehreren Jahren liegen. Wird der Schlüsselwechsel jedoch auf eine fehleranfällige Weise (z. B. Passwortwahl durch Nutzer) durchgeführt, ist in der Praxis ein seltener Wechsel (eines starken Passworts) sicherer als ein häufiger Wechsel (eines schlechteren) [36].

Sicherer Kanal Geheime Schlüssel sollten niemals über ungesicherte Kanäle übertragen werden, damit ein Angreifer nicht in deren Besitz gelangen kann. Existiert zwischen zwei Parteien kein solcher sicherer Kanal, so kann ein sicheres Schlüsselaustauschverfahren (z. B. das Diffie-Hellman Verfahren auf Basis des diskreten Logarithmus) eingesetzt werden um einen symmetrischen Schlüssel auszuhandeln. Dieser kann daraufhin verwendet werden, um einen sicheren Kanal bereitzustellen.

Sicherer Speicher Kryptographische Schlüssel müssen so gespeichert sein, dass das System jederzeit Zugriff auf die Schlüssel hat und diese gleichzeitig vor Angriffen geschützt sind. Dabei müssen geheime Schlüssel so gespeichert sein, dass ein Angreifer keinerlei Zugriff erhalten kann, während öffentliche Schlüssel lediglich schreibgeschützt gespeichert werden müssen. Dadurch wird verhindert, dass ein Angreifer den öffentlichen Schlüssel überschreibt und dadurch fähig ist, scheinbar authentifizierte Nachrichten an das System zu übermitteln.
Idealerweise sind Schlüssel nicht dauerhaft gespeichert - also nicht in persistentem Speicher (engl. Non-Volatile Memory, NVM) - sondern werden nur für die Dauer der Nutzung erzeugt, um das Zeitfenster eines eventuellen Angriffs so klein wie möglich zu halten. Diesen Ansatz verfolgen eine ganze Reihe von Arbeiten unter der Verwendung von *Physikalisch Unklonbaren Funktionen* (PUF). Ein Beispiel für eine PUF ist z. B. das Startverhalten von SRAM-Zellen [37][38][39], welches einzigartig je hergestellter Zelle ist. Der Grund dafür sind minimale Abweichungen beim Herstellungsprozess, welche zu geringen Unterschieden der Leitfähigkeit der Transistoren führen, was wiederum einen unterschiedlichen Initialwert der Zelle nach sich zieht. Eine Sammlung solcher Initialwerte kann etwa als eindeutiger Geräteschlüssel verwendet werden, der nirgends dauerhaft gespeichert ist und nur bei Bedarf (aus dem Startverhalten) erzeugt wird [39].

2.3.2.3 Perfect Forward Secrecy

Perfect Forward Secrecy ist eine Eigenschaft von kryptographischen Protokollen, welche die Auswirkungen der Kompromittierung des Langzeitschlüssels eines Teilnehmers beschreibt. Protokolle die diese Eigenschaft erfüllen, garantieren die Geheimhaltung von zuvor genutzten Sitzungsschlüsseln - und jeglicher damit verschlüsselten Kommunikation - trotz Kompromittierung des Langzeitschlüssels.
Schlüsselaustauschprotokolle auf Basis des Diffie-Hellman Verfahrens (DH) erfüllen unter bestimmten Umständen die *Perfect Forward Secrecy* Eigenschaft.
Diese Bedingung ist, dass für jede Sitzung die geheimen Werte zur Ermittlung des Sitzungsschlüssels zufällig und einmalig gewählt und nach Beendigung der Sitzung zusammen mit dem Sitzungsschlüssel gelöscht werden. Diese Variation des Diffie-Hellman Verfahrens wird als *ephemeral* (vergänglich) bezeichnet und zumeist mit DHE abgekürzt. Dabei wird der private Schlüssel der Teilnehmer lediglich zum Signieren der Schlüsselaustauschparameter genutzt und nicht für die Berechnung des tatsächlichen Sitzungsschlüssels.

2.3.2.4 Zufallszahlengenerator

In vielen kryptographischen Anwendungen werden Zufallszahlen verwendet, z.B. als Einmalzahl (engl. nonce), sicheres Passwort oder als sog. *Salts* um die Entropie (und damit Sicherheit) von Hashverfahren zu erhöhen. Für die Erzeugung von Zufallszahlen werden sogenannte Zufallszahlengeneratoren (engl. Random Number Generator, RNG) verwendet. Diese werden in echte Zufallszahlengeneratoren (engl. True Random Number Generator, TRNG) und Pseudozufallszahlengeneratoren (engl. Pseudo Random Number Generator, PRNG) unterschieden.

TRNGs sind Hardwarekomponenten, die in der Lage sind Zufallszahlen aus zufälligen physikalischen Ereignissen (engl. Noise) zu generieren. Solche physikalische Vorgänge sind z.B. die kosmische Hintergrundstrahlung, Wärmerauschen eines elektrischen Widerstands, oder PUFs (z. B. Startwerte einer SRAM Zelle oder Ringoszillatoren)[37][38]. Im Vergleich zu PRNGs sind hardwarebasierte RNGs jedoch oftmals in ihrer Geschwindigkeit eingeschränkt, d. h. die Erzeugung großer Zufallszahlen benötigt eine lange Zeit, da die Quelle des natürlichen Zufalls beschränkt ist.

PRNGs dagegen basieren auf deterministischen Algorithmen, welche ausgehend von einem ursprünglichen Wert (auch Saat, engl. Seed), scheinbar zufällige Zahlenfolgen erzeugen. Ist der initiale Wert und das verwendete Verfahren bekannt, können die Werte eines PRNGs im Voraus berechnet werden, weshalb die Geheimhaltung der Saat essentiell ist. Erfüllt ein PRNG eine Reihe von Voraussetzungen und Tests so gilt dieser als kryptographisch sicher (engl. Cryptographically Secure PRNG, CSPRNG), solange die Saat dem Angreifer unbekannt ist. Bedingungen und Tests sind im NIST (engl. National Institute of Standards and Technology) Standard 800-90A [40] gegeben. Eine der vorgeschlagenen Methode einen CSPRNG zu implementieren ist die Nutzung einer als sicherer erachteten Blockchiffre (z.B. AES) im Zählerbetriebsmodus (engl. Counter Mode).

2.3.2.5 Hashverfahren

Hashverfahren (oder Streuwertfunktionen, H) bilden eine große Eingabe (engl. Message, M) auf einen kleineren Wert (engl. Digest, D) ab [41]. Hashverfahren welche die folgenden Bedingungen erfüllen werden als kryptographische Hashfunktionen bezeichnet [35].

- **Einwegfunktion:**
  Es ist praktisch unmöglich für gegebenes D ein M zu berechnen sodass gilt $H(M) = D$.
- **2nd-Preimage Eigenschaft:**
  Für einen *gegebenen* Wert M ist es praktisch unmöglich einen gewünschten Wert $M' \neq M$ zu finden, sodass $H(M) = H(M')$ gilt.
- **Starke Kollisionsresistenz:**
  Es ist praktisch unmöglich zwei *beliebige* Werte M und M' (mit $M' \neq M$) zu finden, sodass $H(M) = H(M')$ gilt.

Kryptographische Hashverfahren werden eingesetzt, um Daten auf deren Integrität und Korrektheit zu prüfen, Passwörter nicht im Klartext abzuspeichern oder im Rahmen von digitalen Signaturverfahren. Für den Integritätscheck von gegebenen Daten wird deren Hashwert berechnet und den Daten angehängt oder auf anderem Wege (z. B. einer vertrauenswürdigen Internetseite) veröffentlicht. Der Empfänger kann ebenfalls einen Hashwert der Nachricht berechnen und diesen mit dem übertragenen Hashwert vergleichen. Sind die beiden Werte nicht identisch, so ist die Nachricht fehlerhaft oder manipuliert. Die Integrität der Nachricht kann jedoch nicht sichergestellt werden, sollte ein Angreifer in der Lage sein die Nachricht zu verändern und zusätzlich den Hashwert der veränderten Nachricht zu berechnen und diesen anstatt des eigentlichen Hashwertes zu übermitteln. Der (aktuell) wichtigste Vertreter kryptographischer Hashverfahren ist die SHA (engl. Secure Hash Algorithm) Gruppe, insbesondere in den Versionen SHA-256, SHA-384 und SHA-3.

2.3.2.6 Authentifizierung

Um sicherzustellen dass eine empfangene Nachricht nicht von einem Angreifer gesendet oder verändert wurde, reicht ein einfacher Integritätscheck in der Regel nicht aus. Folglich wird eine zusätzliche Information benötigt, welche den Absender der Nachricht eindeutig identifiziert (engl. Source Authentication). Ähnlich zu den zwei vorgestellten Verfahren der Verschlüsselung von Nachrichten, gibt es im Wesentlichen zwei Techniken zur Authentizitätsprüfung von Nachrichten: ein symmetrisches Verfahren (engl. Message Authentication Code, MAC) und ein asymmetrisches Verfahren (digitale Signatur). Unabhängig von der verwendeten Methode, wird das Signieren einer Nachricht m durch $\mathit{Sign}(m)$ notiert, das Verifizieren der Authentizität mit $\mathit{Ver}(m)$.

Für die Erzeugung symmetrischer MACs können entweder schlüsselbasierte, kryptographisch sichere Hashfunktionen (engl. Keyed-Hash MAC, HMAC) [42] oder Blockchiffren (engl. Cipher-based MAC, CMAC) [43] eingesetzt werden. In beiden Fällen benötigen Sender und Empfänger einen geteilten (symmetrischen) Schlüssel k. Das Ergebnis einer solchen Berechnung wird zumeist als *TAG* der Nachricht bezeichnet. Durch die Nutzung des geheimen Schlüssels für die Berechnung des TAG kann ein Angreifer diesen nicht selbst berechnen. Folglich kann ein Angreifer keine Änderungen an einer Nachricht vornehmen, welche der Empfänger nicht durch eine Prüfung des TAG feststellen kann. Der Besitz des Schlüssels ist somit der Garant für die Authentizität der Nachrichten.
Dies bedeutet insbesondere, dass die Anwendung des symmetrischen MAC-Verfahrens auf eine Gruppe von Teilnehmern das Authentifizierungskriterium aufweicht. Da die Kenntnis des Schlüssels die Authentizität garantiert, kann in einer Gruppe lediglich sichergestellt werden, dass die Nachricht von einem Mitglied der Gruppe gesendet wurde, nicht jedoch von welchem Mitglied.
Beide Verfahren werden als sicher eingestuft [35], solange kryptographisch sichere Verfahren und Schlüssel (z. B. AES-128 Bit) eingesetzt werden. Zudem sollte der TAG nicht erheblich verkürzt (engl. Truncated) werden und zumindest eine Länge von 96-Bit aufweisen [35].

Digitale Signaturverfahren arbeiten auf Basis von Public-Key Verschlüsselungstechniken, im direkten Vergleich wird jedoch die Nutzung des öffentlichen und privaten Schlüssels vertauscht. Der Absender berechnet entsprechend Gleichung (2.1) einen *TAG* zu seiner Nachricht, indem er zunächst einen Hashwert der Nachricht bildet und diesen dann mit seinem privaten Schlüssel verschlüsselt. Nachricht und Signatur können dann an den Empfänger übertragen werden.

$$\textit{Sign}(m) = \{H(m)\}_k^a = \text{TAG} \tag{2.1}$$

TAG: Die Signatur einer Nachricht.

H(): Die Anwendung einer kryptographische Hashfunktion.

$\{x\}_k^a$: Die Anwendung einer asymmetrischen Verschlüsselung mit dem privaten Schlüssel k.

Der Empfänger kann den vom Absender berechneten Hashwert der ursprünglichen Nachricht m mit dem öffentlichen Schlüssel des Absenders entschlüsseln und diesen mit dem neu berechneten Hashwert der empfangenen Nachricht m' vergleichen (vgl. Gleichung 2.2).

$$\textit{Ver}(m', \textit{TAG}) = \begin{cases} \textit{Valid}, & \text{für } \{\text{TAG}\}_{k^{-1}}^a = H(m') \\ \textit{Invalid}, & \text{sonst} \end{cases} \tag{2.2}$$

$\{x\}_{k^{-1}}^a$: Die Anwendung einer asymmetrischen Entschlüsselung mit dem öffentlichen Schlüssel k^{-1} des Senders.

Signaturverfahren können im Gegensatz zu symmetrischen Verfahren auch auf Gruppenkommunikation überführt werden, ohne dass das Authentifizierungskriterium aufgeweicht wird. Solange der private Schlüssel lediglich dem Sender bekannt ist, kann eine digitale Signatur jederzeit diesem Sender eindeutig zugeordnet werden - unabhängig von der Gruppengröße.

Wie bereits im Abschnitt 2.3.2.1 beschrieben, erfordern asymmetrische Verfahren erheblich größere Schlüssel und Rechenaufwand im Vergleich zu symmetrischen Verfahren, weshalb sie selten zur Authentifizierung von Nachrichten eingesetzt werden. Insbesondere für das Authentifizieren von Dateien (z. B. Programmcode) sind digitale Signaturen jedoch weit verbreitet.

2.3.2.7 Public Key Infrastructure

Kernelement asymmetrischer Verfahren ist der sichere Austausch von öffentlichen Schlüsseln, sodass ein Angreifer keine eigenen Schlüssel in das System einbringen kann. Steht kein sicherer Kanal zu deren Übertragung zur Verfügung, so muss der öffentliche Schlüssel wiederum durch ein digitales Zertifikat einer vertrauenswürdigen Quelle zertifiziert sein.
Dadurch ergibt sich eine Kette von (hierarchisch geordneten) Zertifikaten, welche vom Empfänger des öffentlichen Schlüssels geprüft werden kann. Die Kette endet mit einem sogenannten *Root*-Zertifikat, welche von einer *Certificate Authority* (CA) ausgestellt wird. Vertraut der Empfänger der CA (oder einem beliebigen Kettenelement) und sind alle Zertifikate entlang der Kette validiert, so kann der empfangene Schlüssel sicher verwendet werden. Ein solches System wird als *Public Key Infrastructure* (PKI) bezeichnet.
Neben den Signaturen und Schlüsselinformationen enthalten Zertifikate einer PKI zumeist noch weitere Zusatzinformationen. Verbreitet sind Angaben zu Aussteller, Name des Zertifikatsinhabers, Gültigkeitsdauer und Anwendungszweck. Für PKIs ist der *X.509* Standard für digitale Zertifikate (Standardisiert als ISO/IEC 9594-8) weit verbreitet [44].
Eine Erweiterung des Standards stellen die sogenannten Attributzertifikate dar [45]. Diese ähneln im Aufbau den digitalen Zertifikaten, enthalten jedoch keinen (signierten) öffentlichen Schlüssel. Stattdessen sind eine Reihe von Attributen enthalten, welche die Rechte des Zertifikatsinhabers für die Gültigkeitsdauer und Anwendungsbereich des Attributzertifikats beschreiben. Um Manipulationen seitens des Nutzers zu verhindern, werden diese Zertfikate mit dem privaten Schlüssel des Herausgebers signiert und implizit an das Nutzerzertifikat gebunden. Da eine Manipulation ausgeschlossen ist - solange der private Schlüssel des Herausgebers geheim bleibt - können Attributzertifikate eingesetzt werden, um die Rechteverwaltung auf den Nutzer zu verlagern und sind damit ein Möglichkeit die Autorisierung eines Vorgangs zu prüfen.

Als einfaches Beispiel für die Verwendung von Attributzertifikaten kann die Rechteverwaltung von Apps für ein sicheres Betriebssystem betrachtet werden. Der Hersteller des Betriebssystems stellt vertrauenswürdigen Apps eine digitale Signatur und ein zugehöriges Attributzertifikat aus, welches die Rechte (z. B. Installation, Zugriff auf die Internetverbindung oder Dateizugriff) der entsprechenden App beschreibt.
Fordert die App zu einem späteren Zeitpunkt eine privilegierte Funktion des Betriebssystem an, so kann dieses anhand des Attributzertifikats die Zugriffsrechte der App auf die entsprechende Funktion prüfen. Zu den Vorzügen dieses Ansatzes gehört die dezentrale Rechteverwaltung, wodurch dynamische, verteilte Systeme möglich sind. So muss das Betriebssystem insbesondere nicht sämtliche Zugriffsrechte aller möglichen Apps zentral speichern (oder die entsprechenden

Rechte zunächst vom Hersteller abfragen), sondern lediglich das Attributzertifikat auf dessen Validität und die enthaltenen Rechte prüfen.

2.3.3 Fehlerkorrigierende Codes

Fehlerkorrigierende Codes (engl. Error Correcting Code, ECC) werden eingesetzt, um Daten mit redundanten Informationen zu ergänzen. Diese erlauben es Fehler bei der Speicherung oder Übertragung der Daten zu erkennen und je nach Anzahl der Fehler auch zu korrigieren. Ein zentraler Begriff im Zusammenhang mit ECCs ist der Hammingabstand zweier Codewörter, definiert als die Anzahl der Bitpositionen, welche in den beiden Codewörtern unterschiedlich sind.

Für diese Arbeit sind insbesondere *Erasure*-Codes von Interesse [46]. Für *Erasure*-Codes wird die folgende Notation verwendet:

$$C_{k,r}(X) \rightarrow Y \tag{2.3}$$

X Eine Menge von k Datensymbolen $x_1, ..., x_k$.

r Die Anzahl zusätzlicher Symbole, welche durch den *Erasure*-Code erzeugt werden.

Y Die Menge von $k + r$ Codesymbolen $y_1, ..., y_{k+r}$

Ein solcher Code besitzt die Eigenschaft, dass eine beliebige Teilmenge von Y der Größe k ausreichend ist, um alle ursprünglichen Datensymbole von X wiederherzustellen. Jedes Datensymbol besteht dabei aus b-Bits, wobei reale Erasure-Codes zumeist auf Symbolen der Größe 2^n arbeiten. Ein bekanntes Beispiel für Erasure-Codes sind *Reed-Salomon* [46] und *Tornado*-Codes.

2.3.4 Technische Einschränkungen

Aufgrund des hohen Preisdrucks und der hohen Stückzahlen sind ECUs im Allgemeinen minimalistisch ausgestattet, d. h. die Rechenleistung einer jeden ECU ist in etwa so bemessen, dass diese die vorgesehene Aufgabe erfüllen kann - und nicht viel mehr. Ein zusätzlicher Grund für die geringe Leistungsfähigkeit der meisten ECUs ist die Notwendigkeit zur Minimierung des Stromverbrauchs des Gesamtsystems. Bereits für herkömmliche Fahrzeuge mit Verbrennungsmotoren ist ein geringer Gesamtstromverbrauch wichtig, da der benötigte Strom bereits während dem Start durch die Fahrzeugbatterie erbracht werden muss. Dabei ist der Strombedarf der ECUs während des Systemstarts sogar erhöht, da währenddessen Selbsttests der Hardware durchgeführt werden, um einen korrekten Betrieb zu prüfen. Dieser sogenannte Startstrom muss dabei möglichst gering ausfallen, damit auch bei einem niedrigen Ladestand der Batterie, etwa bei kalten Temperaturen oder langer Standzeit, ein Start möglich ist. Dass dies ein erhebliches Problem darstellt, ist deutlich anhand der jährlichen ADAC Pannenstatistiken zu sehen. Im Jahr 2007 machten Pannen, welche auf einen Defekt oder eine

unzureichend geladene Batterie zurückzuführen waren, über 30% aller Pannen aus [47]. Dieser Anteil stieg kontinuierlich und liegt laut der letzten Statistik aus dem Jahr 2016 bei über 35%, trotz einer insgesamt besseren Qualität der verwendeten Batterien [48]. Dieses Phänomen wird insbesondere auf die gestiegenen Ansprüche an die Batterien durch eine stetig steigende Anzahl verbauter ECUs zurückgeführt [49].
Während der Fahrt muss der benötigte Strom durch die Lichtmaschine des Fahrzeugs bereitgestellt werden, was letztendlich zu einem erhöhten Kraftstoffbedarf führt. Für rein elektrisch betriebene Fahrzeuge ist der Gesamtstromverbrauch noch entscheidender, da der Strom für die ECUs ebenfalls von den Batteriereserven des Fahrzeugs bezogen werden muss und damit die ohnehin beschränkte Reichweite weiter reduziert.

Durch die geringen Leistungsreserven und zugleich hohen Echtzeitanforderungen für sicherheitskritische ECUs (z.B. Airbags und Bremsen) kann der Einsatz rechenintensiver kryptographischer Primitive das Zeitverhalten und damit die Betriebssicherheit der Komponenten negativ beeinflussen. Insbesondere die Anwendung asymmetrischer Verfahren ist auf den meisten ECUs nicht in der erforderlichen Geschwindigkeit möglich.
Neben den eigentlichen Steuergeräten sind auch die verwendeten Bussysteme kaum für eine sichere Kommunikation geeignet. Insbesondere der weit verbreitete CAN-Bus ist aufgrund der geringen Datenrate (in der ursprünglichen Form lediglich 8 Byte Nutzdaten je Datenframe) und des Broadcast-Prinzips praktisch unmöglich abzusichern [23]. Bereits ein einfacher Integritätsschutz mittels eines 128-Bit MAC würde folglich einen Overhead von zwei vollen CAN-Frames erzeugen.
Für die CAN-FD Erweiterung besteht dagegen, aufgrund der erhöhten Nutzdatenlänge, durchaus die Möglichkeit für eine Absicherung. Neben der erhöhten Übertragungsrate wird dies als entscheidender Vorteil für die Zukunftsfähigkeit des CAN-Bus angesehen.

3 SOME/IP

Das SOME/IP Protokoll hat das Ziel, eine einheitliche Middleware für die IP-basierte Kommunikation innerhalb von Fahrzeugen zu definieren [50][51][52]. Das Protokoll setzt auf einen vorhandenen TCP/UDP Stack auf und schafft so eine weitere Abstraktionsebene für die Kommunikation zwischen Applikationen, indem Lokalitätstransparenz geschaffen wird. Diese Eigenschaft bezeichnet die Tatsache, dass eine Anwendung kein Wissen darüber besitzen muss, welcher Netzwerkknoten eine gewünschte Funktion oder eine gesuchte Information bereitstellt. Befindet sich diese auf derselben ECU, wird die Information über lokale Kommunikationskanäle (z. B. Shared-Memory, Interprozesskommunikation, Hypervisor-Schnittstelle) bereitgestellt. Sollte die Information dagegen auf einem anderen (bekannten) Netzwerkknoten vorhanden sein, führt die Middleware die nötige Netzwerkkommunikation durch und übergibt die Daten anschließend an die Applikation.

3.1 Services

Das zentrale Element des SOME/IP Protokolls bilden, gemäß dem serviceorientierten Paradigma, die sogenannten *Services*. Ein Service ist definiert als eine logische Kombination von Methoden, Datenfeldern und Events, welche anderen Services bereitgestellt wird.
Events bezeichnen dabei eine abstrakte Änderung des Servicezustands und erlauben eine effiziente Implementierung signalorientierter Kommunikation ähnlich zum CAN-Protokoll. Wann ein solches Event ausgelöst wird, kann beispielsweise durch die Notifier-Methoden der entsprechenden Felder definiert werden. Ein oder mehrere Events eines Service können zu einer sogenannten *Eventgruppe* zusammengefasst werden. Clients abonnieren eine oder mehrere Eventgruppen eines Services, um die enthaltenen Event-Benachrichtigungen der abonnierten Gruppe zu erhalten. Events lassen sich effizient mit Multicast-Kommunikation implementieren, sollte die Anzahl der abonnierten Clients steigen.

Es gibt drei grundlegende Versionen von Notifier-Methoden, welche von SOME/IP unterstützt werden. Die einfachste Form *Update-on-Change* sendet eine Benachrichtigung sobald sich der Wert des entsprechenden Felds ändert. Sollen nur wesentliche Wertänderungen ein Event auslösen, können sog. *Epsilon-Change* Notifier konfiguriert werden. Dazu wird ein ϵ-Wert für das Feld definiert. Sobald sich der zuletzt gesendete Wert um mehr als das definierte ϵ verändert, wird ein Notify ausgelöst. Als dritte Variante ist ein *Cylcic-Update* möglich. Dazu wird unabhängig von einer möglichen Wertänderung der aktuelle Wert des Felds in festgelegten Zeitintervallen gesendet.

3.1.1 Beispiel-Services

Im Folgenden wird die Beispielarchitektur aus Kapitel 2.1.2 um passende Applikationen (repräsentiert durch deren Services, Eventgruppen und Methoden) erweitert. Diese besitzen eine Reihe von Abhängigkeiten untereinander, welche zur Verdeutlichung der Funktionsweise des Protokolls verwendet werden können. Eine Darstellung der Abhängigkeiten auf ECU-Ebene wird in Abbildung 3.1 veranschaulicht.

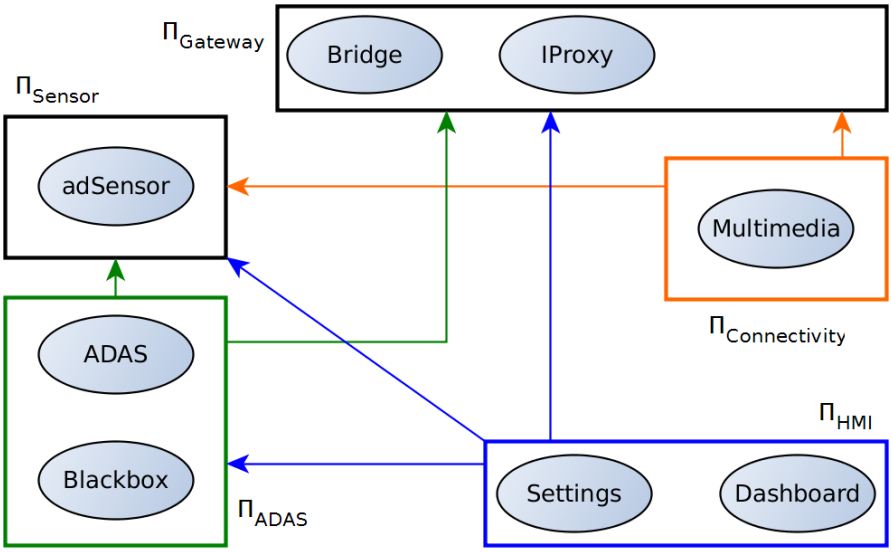


Abbildung 3.1: Darstellung der Abhängigkeiten des Beispielsystems. Eine Abhängigkeit wird durch einen Pfeil repräsentiert. Um die Übersichtlichkeit zu erhalten sind Abhängigkeiten auf ECU-Ebene dargestellt.

ADAS Diese Applikation stellt eine Sammlung von Fahrassistenzsystemen dar. Der Service beinhaltet eine Reihe von Konfigurationsmethoden (*config*), sowie eine Eventgruppe (*warnings*), welche im Fall einer erkannten Gefahr Warnnachrichten sendet.
Um diese Funktionalitäten bereitzustellen benötigt die ADAS-Applikation eine Vielzahl von Sensordaten, welche durch die Eventgruppen (*sensorData* und *CANread*) bereitgestellt werden. Überschreitet die erkannte Gefahr einen Schwellenwert, so reagiert die Applikation zudem selbstständig, um einen Unfall zu verhindern (z. B. durch das Einleiten einer Notbremsung). Dazu muss die ADAS-Applikation in der Lage sein bestimmte CAN-Nachrichten auszulösen, was durch die *CANwrite* Methoden möglich ist. Zusätzlich kann die Übertragung von Warnnachrichten mittels Funkschnittstellen (z. B. C2X) nötig werden, weshalb die *transmit*-Methode des *adSensor*-Service ebenfalls benötigt wird.

adSensor Dieser Service bündelt eine Reihe fortgeschrittener Sensoren und stellt deren Daten durch die Eventgruppe *sensorData* anderen Services zur Verfügung. Insbesondere C2X-Systeme erlauben neben dem rein passiven Sensorenbetrieb auch die Übertragung von

Nachrichten, z. B. um nachfolgende Fahrzeuge vor einem Stau oder einer Notbremsung zu warnen. Für diese Funktionalität bietet der Service die *transmit*-Methode an.

Bridge Der Gateway-Service stellt die Schnittstelle zwischen Ethernet-basierten SOME/IP Netzwerk und dem CAN-Bus der essentiellen Fahrzeugfunktionen bereit. Bestimmte CAN-Nachrichten werden als Events durch die *CANread*-Eventgruppe verfügbar gemacht, während die *CANwrite*-Methoden einen kontrollierten schreibenden Zugriff auf den CAN-Bus gestatten.

IProxy Das Gateway stellt zusätzlich einen Proxy-Dienst für den Internetzugang mittels der Mobilfunkschnittstelle des Fahrzeugs zur Verfügung. Nutzende Services können sich mithilfe der *iRequest*-Methode anmelden und erhalten die notwendigen Zugangsinformation als Antwort.

Multimedia Dieser Service verwendet die *iRequest*-Methode um Multimedianwendungen (z. B. Navigationssystemen oder Online-Suchanfragen) den Internetzugriff zu ermöglichen. Zusätzlich greift dieser auf bestimmte Sensordaten des *adSensor*-Service zu. Dies ist notwendig, um beispielsweise den Videostrom einer Rückfahrkamera anzuzeigen, oder die Wiedergabelautstärke automatisch an die Fahrtgeschwindigkeit anzupassen.

Settings Hierbei handelt es sich um eine Steuerschnittstelle, welche diverse, vom Fahrer getätigte, Einstellungen an das System weiterleitet. Dazu werden die entsprechenden Set-Methoden der betroffenen Service benötigt, z. B. die *config*- und spezielle *CANwrite*-Methoden.

Dashboard Dieser Service zeigt alle wichtigen Fahrzeuginformation und Warnungen in einer leicht zugänglichen Form im Cockpit des Fahrzeugs an. Dazu benötigt der Service eine Vielzahl von Informationsquellen, insbesondere die Eventgruppen *warnings*, *sensorData* und *CANread* sind dabei von Interesse.

Blackbox Im wesentlichen ist der Blackbox-Service ein rein passiver Beobachter der internen Kommunikation, welcher Daten für eine spätere Diagnose im Fall eines Unfalls oder für Diagnosezwecke sammelt. Dazu abonniert der Service sämtliche Eventgruppen des Systems und bietet zudem eine *logMe*-Methode an. Andere Services können diese Methode aufrufen, um ein Logging sämtlicher übertragener Nachrichten des eigenen Service zu aktivieren. Dies ist beispielsweise für einen Service interessant, welcher (teilweises) autonomes Fahren gestattet. Kommt es zu einem Unfall, kann die Gesamtheit der aufgezeichneten Kommunikation Aufschluss über die Unfallursache geben. Durch eine Internetanbindung mithilfe der *iRequest*-Methode können beispielsweise (anonymisierte) Diagnosedaten des Services an den Hersteller übermittelt werden.

Tabelle 3.1 zeigt eine Übersicht über die zuvor beschriebenen Services, deren jeweiligen angebotenen Methoden und Eventgruppen sowie den Abhängigkeiten zu anderen Services. Die letzte Spalte gibt Aufschluss über die ECU auf welcher der Service alloziert ist.

ID	Service	Methoden	Eventgruppen	Benötigt	Knoten
1	ADAS	config	warnings	transmit, sensorData, CANread, CANwrite	Π_{ADAS}
2	adSensor	transmit	sensorData	-	Π_{Sensor}
3	Bridge	CANwrite	CANread	-	$\Pi_{Gateway}$
4	IProxy	iRequest	-	-	$\Pi_{Gateway}$
5	Multimedia	-	-	sensorData, iRequest	$\Pi_{Connectivity}$
6	Settings	-	-	config, CANwrite	Π_{HMI}
7	Dashboard	-	-	warnings, CANread, sensorData	Π_{HMI}
8	Blackbox	logMe	-	alle Eventgruppen, iRequest	Π_{HMI}

Tabelle 3.1: Übersicht über einen Satz beispielhafter SOME/IP Services und deren Abhängigkeiten.

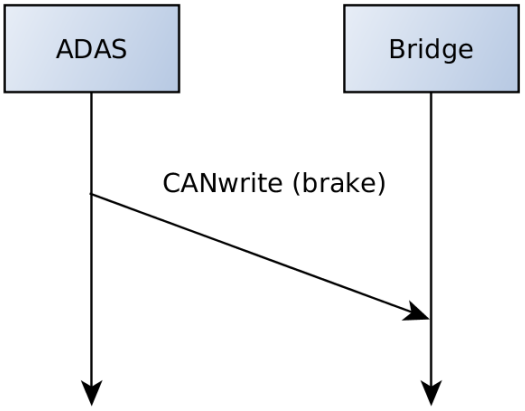
3.1.2 SOME/IP Kommunikationskonzepte

Der SOME/IP Standard definiert vier zentrale Kommunikationskonzepte.

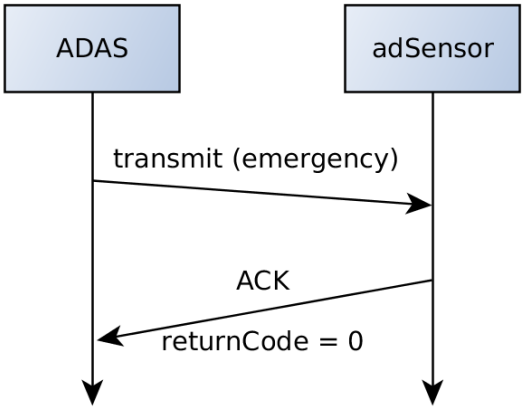
RPCs Entfernte Methodenaufrufe (engl. remote proccedure calls, RPCs) werden in *Fire&Forget* und *Request-Response* unterschieden. Bei der ersten Variante ruft der Client eine vom Server angebotene Methode auf, erwartet jedoch keinen Rückgabewert (vgl. Abbildung 3.2a). *Fire&Forget* Anfragen sind folglich insbesondere für Steuerbefehle geeignet. Die *Request-Response* Variante ist die klassische Form eines entfernten Methodenaufrufs und der aufrufende Client erhält eine Antwortnachricht, welche einen *returnCode* und eventuelle Rückgabewerte enthält (vgl. Abbildung 3.2b).

Felder Datenfeldern eines Service können durch Clients mithilfe von *Get-* bzw. *Set*-Methoden ausgelesen und geändert werden (vgl. Abb. 3.2c). Zudem kann jedes Feld eine Notifier-Methode besitzen, welche den aktuellen Wert des Felds an interessierte Clients überträgt.

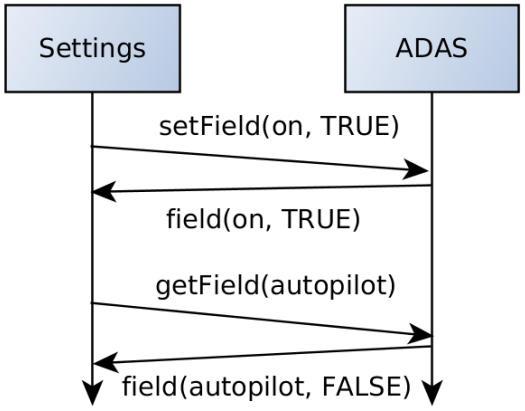
Publish-Subscribe Ein Service kann eine oder mehrere Eventgruppen anbieten, welche von interessierten Clients abonniert werden können. Sobald sich der Zustand der zur Eventgruppe gehörenden Felder, in einer definierten Weise ändert, sendet der Service ein Benachrichtigungsframe (Event) mit dem neuen Zustand an alle abonnierenden Clients (vgl. Abbildung 3.2d). Durch diesen Mechanismus lässt sich signalorientierte Kommunikation effizient implementieren.



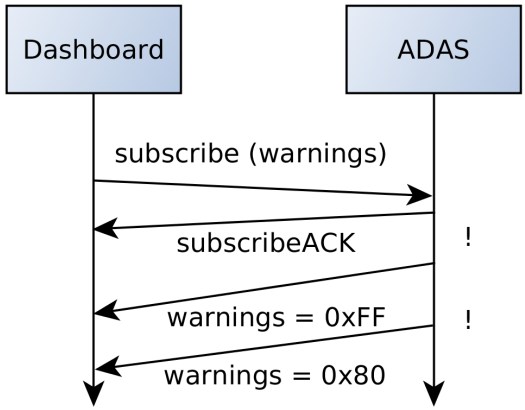
(a) "Fire&Forget"-RPC: Methodenaufruf ohne Antwort.



(b) Request-Response-RPC: Methodenaufruf mit Antwortnachricht.



(c) Felder: Setzen bzw. Auslesen der Datenfelder eines anderen Service.



(d) Publish-Subscribe: Client abonnieren eine angebotene Eventgruppe des Servers und erhält fortan Benachrichtigungen über Events.

Abbildung 3.2: Die vier zentralen SOME/IP Kommunikationsparadigmen.

3.2 Identifier

Ein Service besitzt eine innerhalb des Systems eindeutige 16-Bit lange *Service-ID*. Zusammen mit der IP-Adresse und dem Port des Service kann ein Service eindeutig identifiziert werden. Mehrere Instanzen eines (funktional identischen) Services sind möglich und werden durch eine ebenfalls 16-Bit lange *Service-Instanz-ID* eindeutig identifiziert, wobei die reservierte Instanz-ID *0xFFFF* eine Adressierung aller Instanzen erlaubt. Um den SOME/IP-Header möglichst klein zu halten, wird in diesem lediglich die Service-ID angegeben. Die Service-Instanz-ID wird nur während der SOME/IP Service Discovery (siehe Abschnitt 3.5) genutzt. Als Folge davon dürfen zwei Instanzen eines Service, welche

auf derselben ECU platziert sind, nicht auf demselben TCP/UDP Port empfangen und senden, um eine falsche Adressierung auszuschließen.

Jede Methode und Event-Notification, welche ein Service anbietet, besitzt eine ebenfalls 16-Bit große *Method-ID*, welche die Methode innerhalb des Service-Kontextes eindeutig identifiziert. Per Konvention ist das höchstwertige Bit (engl. most significant bit, MSB) für Events '1' und für Methoden '0'. Eventgruppen dagegen besitzen systemweit eindeutige 16-Bit IDs.

Die Konkatenation von Service-ID und Method-ID identifiziert eindeutig den Empfänger einer SOME/IP Nachricht und wird deshalb als *Message-ID* bezeichnet. Die Message-ID - in Verbindung mit IP-Adresse und der Portnummer - reicht aus um die Nachricht an den korrekten Server zu senden.

3.3 Session Handling

Zur Unterscheidung unterschiedlicher Client-Anfragen wird eine Form des Session Handling zwischen Client und Server benötigt. SOME/IP sieht dazu die Verwendung der *Request-ID* vor, welche sich aus den beiden 16-Bit Felder *Client-ID* und *Session-ID* zusammensetzt.

Client-ID Dieser Identifier bezeichnet die Client-Applikation innerhalb einer ECU eindeutig. Die Zuordnung einer ID zu einer Applikation kann von der ECU frei verwaltet werden.

Session-ID Im einfachsten Fall ist dieser Identifier ein einfacher Zähler, der bei jeder neuen Anfrage des Clients inkrementiert wird um eine Unterscheidung zu jeder vorangegangenen Anfrage zu ermöglichen. Der SOME/IP-Standard sieht vor, dass für die erste Nachricht des Clients an den Server die Session-ID den Wert *0x01* annimmt.

Session Handling ist für RPCs mit Antwort zwingend erforderlich und optional für "Fire&Forget"-RPCs und Events.

3.4 SOME/IP Frames

Das SOME/IP Protokoll definiert einen kurzen Header zur korrekten Verarbeitung der Nachricht. Abbildung 3.3 zeigt die Felder des 128-Bit großen Headers. Die Message-ID bezeichnet den Empfänger des Frames (vgl. 3.2), die Request-ID den Absender und die aktuelle Session (vgl. 3.3). Das Längenfeld beschreibt die gesamte Länge der SOME/IP Nachricht. Dies ist insbesondere erforderlich, da mehrere SOME/IP Nachrichten an

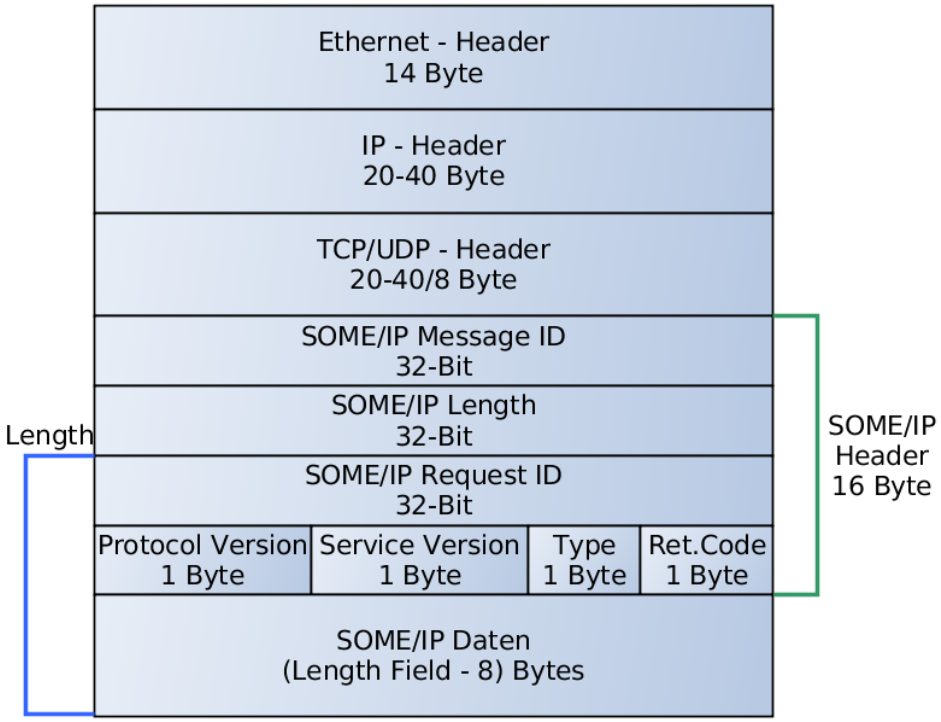


Abbildung 3.3: SOME/IP Header.

denselben Empfänger konkateniert werden können, um die Übertragungsrate zu erhöhen. Ein Empfänger kann diese unterscheiden, indem er die im Header angegebene Länger mit der tatsächlichen Länge des Frames vergleicht. Ist diese größer als angegeben, folgt ein weiterer SOME/IP Frame. Im letzten 32-Bit Feld des Headers folgen vier Bytefelder für Protokollversion, Service Version, Nachrichtentyp und Rückgabewert. Protokollversion bezeichnet die SOME/IP Version (aktuell Version 0x01), wohingegen die Service-Version eine vom Server spezifizierte Version des angebotenen Service bezeichnet, um veraltete Service-Definitionen zu erkennen. Der Nachrichtentyp lässt sich auf die in 3.2 vorgestellten Kommunikationstypen abbilden. Zudem wird zwischen Anfrage-, Antwort-, Bestätigungs- und Fehlerframes unterschieden. Das Rückgabefeld ist zur Vereinfachung des Header-Layouts in jedem Frame enthalten, wird jedoch nur in Antwort- und Fehlerframes verwendet.

Die Länge der SOME/IP Nutzdaten wird durch das zugrundeliegende Transportprotokoll (UDP oder TCP) sowie der Ethernet-Fragmentierung beschränkt, welche vermieden werden sollte. Bei der Verwendung von UDP über Standardethernet und IPv4, ist ein SOME/IP Frame (inklusive Header) auf eine maximale Größe von 1472 Bytes beschränkt, bevor Paketfragmentierung einsetzt. Das SOME/IP Protokoll schreibt eine maximale Länge der Nutzlast von 1400 Bytes vor. Die verbleibenden 56 Byte ($1472 - 1400$ Byte Nutzdaten -16 Byte Header $= 56$) sind für zusätzliche Header und Protokolldaten reserviert. Das aktuelle SOME/IP Protokoll sieht keinerlei Security Maßnahmen vor, die

56 Byte reservierte Nutzdaten sind jedoch explizit für mögliche Security Erweiterungen angedacht. Solange ein Sicherheitsprotokoll nicht mehr als diese 56 Bytes an zusätzlichen Daten pro Frame übertragt, kann somit eine Absicherung der SOME/IP Kommunikation ohne Änderung des Standards und kompatibler Implementierungen erfolgen.

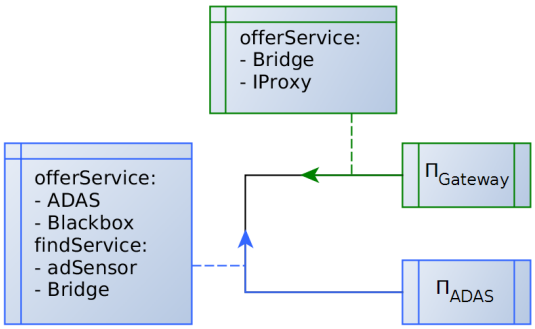
3.5 Service Discovery

Voraussetzung für die SOME/IP Middleware ist das Wissen über angebotene Services, deren aktuellen Status und Position innerhalb des Systems. Für diese zentrale Aufgabe definiert SOME/IP das *Service Discovery* (SD) Protokoll. Das Auffinden von Services erfolgt dabei durch den SOME/IP-SD Prozess einer ECU und unterteilt sich in die lokale (für alle Services die auf derselben ECU ausgeführt werden) und systemweite (für Services die auf ECUs innerhalb des Netzwerks verfügbar sind) Service Discovery.
Sobald eine ECU die lokale Service Discovery abgeschlossen hat sendet diese eine zusammengesetzte SOME/IP-SD Nachricht als IP-Multicast an alle weiteren ECUs. Eine solche Nachricht enthält eine oder mehrere *offerService* und *findService* Teilnachrichten. Eine *offerService*-Nachricht beschreibt einen lokalen Service der ECU, inklusive aller zugehörigen Optionen. Alle Empfänger einer solchen Nachricht können die so angebotenen Service mithilfe von SOME/IP einzusetzen.
Kann die Abhängigkeit eines Service nicht lokal aufgelöst werden, so wird eine entsprechende *findService*-Nachricht übertragen. Ein Empfänger, der eine solche Teilnachricht liest, überprüft daraufhin seine lokalen Services auf eine Übereinstimmung. Ist eine gefunden antwortet der Empfänger mit einer erneuten SOME/IP-SD Nachricht, welche den *offerService* Eintrag für den gesuchten Service enthält.
Abbildung 3.4 zeigt den Ablauf einer beispielhaften Service Discovery.

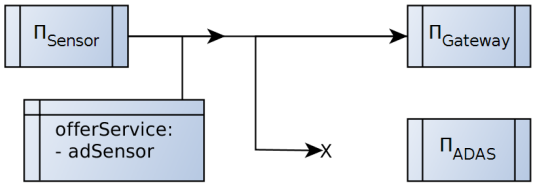
3.5.1 Systemstart

Nach dem Start des SOME/IP-SD Prozesses, durchläuft dieser drei Phasen: *Initial Wait*, *Repetition* und *Main*.

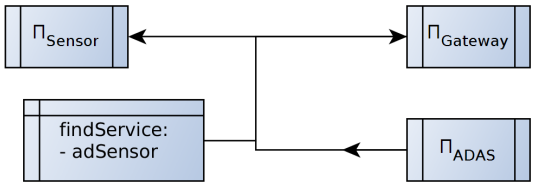
Initial Wait Nach dem Systemstart tritt der Prozess in eine Wartephase ein, bevor die eigentliche Service Discovery beginnt. Diese Warteperiode dient dazu die lokale Service Discovery abzuschließen und langsamen Systemen ebenfalls den Start und die anschließende Teilnahme am SOME/IP-SD Protokoll zu ermöglichen. Im besten Fall sind somit alle SOME/IP-SD-Prozesse im Netzwerks bereit und es genügt der Austausch eines einzigen SOME/IP-SD Nachrichtensatzes je ECU, um alle Services im Netzwerk zu entdecken.



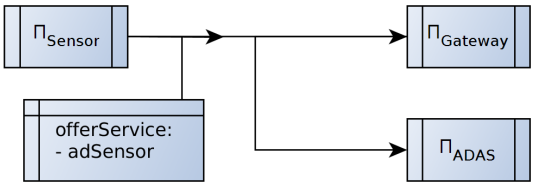
(a) $\Pi_{Gateway}$ und Π_{ADAS} übertragen ihre jeweiligen SOME/IP-SD Nachrichten.



(b) Π_{Sensor} startet verspätet und sendet die eigene SOME/IP-SD Nachrichten. Diese erreicht jedoch nur $\Pi_{Gateway}$.



(c) Aufgrund des vorherigen Paketverlustes hat Π_{ADAS} die SOME/IP-SD Nachricht von Π_{Sensor} nicht erhalten und sucht weiterhin nach dem *adSensor*-Service.



(d) Π_{Sensor} hat die zuvor übertragene Suchanfrage erhalten und erkennt, dass ein lokaler Service gesucht wird und antwortet mit einer erneuten *offerService*-Nachricht.

Abbildung 3.4: SOME/IP Service Discovery am Beispiel dreier ECUs.

Repetition Nach Ablauf der Wartephase, beginnen SOME/IP-SD Prozesse die zusammengesetzten Nachrichten per IP-Multicast an eine vorkonfigurierte Multicast-Adresse zu senden (vgl. Teilbild 3.4a). Diese werden in regelmäßigen Intervallen wiederholt. Die Intervall- und Gesamtlänge dieser Phase ist ein Systemparameter und kann beliebig gewählt werden. Eine typische Wahl für diese Parameter ist z. B. eine Gesamtdauer von $40\,ms$ bei einer Intervalllänge von $10\,ms$.

Main An die Wiederholungsphase schließt sich die Hauptphase an, welche bis zur Ende der Laufzeit der ECU anhält. In dieser Phase beantwortet der SD-Prozess eingehende *findService*-Anfragen, sollte zumindest ein lokaler Service einem Eintrag der Anfrage entsprechen (vgl. Teilbild 3.4c und 3.4d). Zudem kann in regelmäßigen Abständen ein *cyclic update* der SOME/IP-SD Nachricht gesendet werden, z. B. um die Gültigkeitszeit (engl. time to live, TTL) der angebotenen Services zu erneuern.

3.5.2 SOME/IP-SD Nachrichten

Das Service Discovery Protokoll definiert einen komplexen zusammengesetzten Nachrichtentyp zum Anbieten bzw. Auffinden von Services und der Übertragung entsprechender Optionen (u.a. IP-Adresse, Port und Protokoll). Abbildung 3.5 zeigt den allgemeinen Aufbau einer SOME/IP-SD Nachricht. Jede Nachricht enthält eine Reihe von *Entries* und

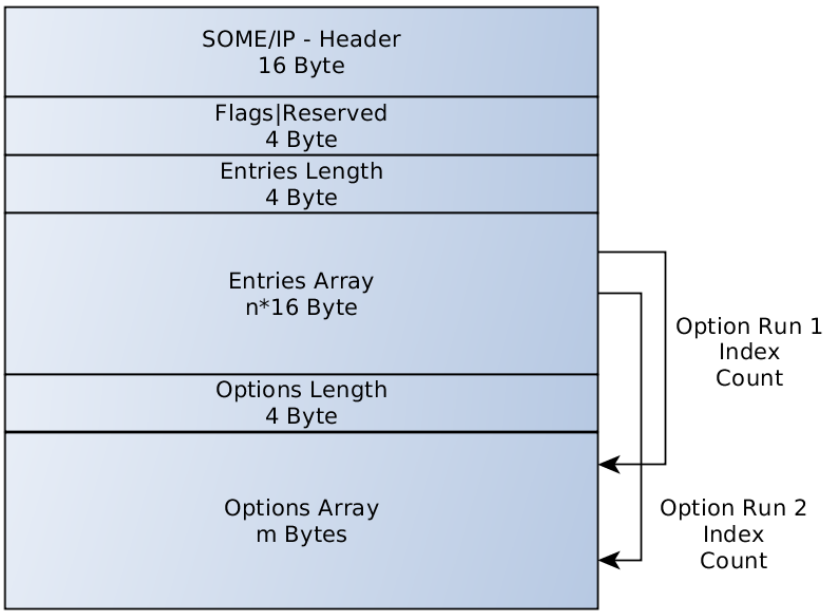


Abbildung 3.5: SOME/IP Service Discovery Nachricht. Die zweistufige Optionssuche ermöglicht eine Reduktion von Redundanz gleicher Optionen durch Referenzierung auf gemeinsame Optionen verschiedener Einträge.

zugehöriger Optionen.

Entry Ein Entry beschreibt entweder genau einen Service oder eine Eventgruppe, durch die Angabe der zugehörigen Service-ID und Service-Instanz-ID. Weiter enthält jeder Eintrag eine Service-Version und ein TTL-Feld, welches die Gültigkeit des Eintrags (in Sekunden) angibt. Zudem bezeichnet jeder Eintrag, ob es sich um einen *findService* oder *offerService* Eintrag handelt - dies erlaubt das Zusammenfassen der beiden Typen in eine große SOME/IP-SD Nachricht. Der einzige Unterschied zwischen Eventgruppen und Service Einträgen findet sich im letzten 32-Bit Feld des Eintrags. Serviceeinträge übertragen die Minor-Version des Service, während Eventgruppeneinträge die 16-Bit Eventgruppen-ID übertragen (die verbleibenden 16-Bit werden aktuell nicht genutzt).

Optionssuche Einträge enthalten zusätzlich die notwendigen Information zum Auffinden der zugehörigen Optionen innerhalb des *Options*-Feldes. SOME/IP-SD definiert dazu eine zweistufige Optionssuche (*Option Run 1* und *Option Run 2*). Für jeden *Run* speichert der Eintrag den Startindex innerhalb des Optionsfelds und die Anzahl der Optionen, die ab dieser Position gelesen werden sollen (vgl. Abbildung 3.5). Dies ist möglich, da jede Option ein eigenes Längenfeld besitzt, wodurch einzelne Optionen eindeutig voneinander unterschieden werden können. Zusammen ermöglicht dies den Einsatz spezifischer Optionen einerseits, sowie Referenzierung auf gemeinsam genutzte Optionen. Durch die Wiederverwendung häufig verwendeter Optionen

(etwa die IP-Konfiguration des anbietenden Servers und die des SOME/IP-SD Programms) kann die Redundanz der übertragenen Nachricht erheblich reduziert und damit die Effizienz des SOME/IP-SD Verfahrens erhöht werden.

3.5.3 Lokale Service Discovery

Zur Erstellung der SOME/IP-SD Nachricht muss der Prozess genaue Informationen über alle Services besitzen, welche lokal auf der ECU ausgeführt werden. Dieses Wissen wird benötigt, um die *offerService* und *findService* Einträge der Nachricht erstellen zu können. Die genaue Implementierung der Informationsermittlung ist dabei vom SOME/IP Standard nicht genauer festgelegt.
Für AUTOSAR erfolgt dies - je nach Plattform - wie folgt:

AUTOSAR classic Durch Analyse der (statischen) ECU-Konfigurationsdatei nach Abschluss der ECU-Allokation können die benötigten Informationen extrahiert werden, da alle Methoden, Variablen und Datentypen der Anwendungen nach Abschluss der Erzeugung statisch und bekannt sind. Die Konfigurationsdatei liegt lokal auf der ECU und kann folglich von dem SOME/IP-SD Prozess ausgelesen werden.

Adaptive AUTOSAR Für jede Anwendung, welche einen Service anbietet oder nutzt, wird im Rahmen der Generierung eine *Service Interface Definition* Beschreibung erzeugt - im Folgenden einfach: *Manifest* genannt. Diese enthält alle nötigen Informationen für die SOME/IP Service Discovery (u.a. die Service Version, die *Service-*, *Method-* und *Eventgruppen*-Identifier) und für die Serialisierung von Datenübertragungen (z. B. Beschreibung von RPC-Parametern und deren Datentypen). Die Definitionsdatei ist - zusammen mit dem Programm der Anwendung - lokal auf der ECU gespeichert und kann folglich von dem Service Discovery Prozess analysiert werden. Somit kann die Menge aller angebotenen und benötigten Services der lokalen ECU erstellt werden.

Die lokale Service Discovery stellt zudem sicher, dass alle Services innerhalb einer ECU jederzeit für andere lokale Services zur Verfügung stehen und die SOME/IP Middleware keinen *Single Point of Failure* aufweist, was im Falle einer zentralen Service-Verwaltung der Fall wäre.

3.6 Skalierbarkeit durch interoperable Ausprägungen

Besonderes Augenmerk von SOME/IP ist auf die Skalierbarkeit des Protokolls gelegt, damit sowohl leistungsschwache als auch leistungsstarke ECUs über dasselbe Protokoll kommunizieren können. Um die Skalierbarkeit zu erreichen sind mehrere interoperable Ausprägungen von SOME/IP möglich.

Für eine Minimalausprägung kann z. B. das gesamte SOME/IP-SD Protokoll durch eine statische Konfigurationsdatei ersetzt werden, welche alle notwendigen Informationen enthält. Zudem kann auf die Implementierung von TCP verzichtet werden. Für die Interoperabilität mit einer aufwändigeren Implementierung ist lediglich die SOME/IP Serialisierung und die Unterstützung der in Abschnitt 3.1 vorgestellten Kommunikationstypen notwendig.

4 Sicherheitsanalyse SOME/IP

Die Betriebssicherheit von Kraftfahrzeugen ist ein wesentlicher Fokus von Automobilherstellern und wird, seit dessen Veröffentlichung, stark von der ISO Norm 26262 geprägt [53]. Die Anwendung und Einhaltung des Standards ist grundsätzlich freiwillig, jedoch sind Hersteller nach deutschem Recht verpflichtet der generellen Produkthaftung, sowie der sogenannten *Verkehrssicherungspflicht* nachzukommen [54]. Diese fordert von Herstellern, dass Produkte den zum Zeitpunkt der Herstellung erwartbaren Sicherheitsanforderungen genügen, wobei Verstöße mit Schadensersatzforderungen geahndet werden können. Geltende Normen werden daher verbreitet als Hilfsmittel für die Sicherung der Produktqualität betrachtet und unter Mitwirkung der Hersteller stets weiterentwickelt [55].
Kapitel 3 der Norm beschreibt die Anwendung einer klar definierten Methodik zur Gefährdungs- und Risikoanalyse. Anhand dieser Analyse lassen sich einzelne Komponenten oder Funktionen des Gesamtsystems bestimmten *ASIL* (engl. Automotive Safety Integrity Level) zuordnen. Je höher das ASIL-Level einer Komponente bzw. einer Funktion, desto höher sind die gestellten Anforderungen an die Implementierung, Tests und eingesetzte Hardware.
Die Informationssicherheit des betreffenden Systems wird in der ISO-26262 Norm jedoch nicht speziell betrachtet. Um diese Lücke zu schließen, wird aktuell die ISO-Norm 21434 spezifiziert, welche speziell den Bereich der Informationssicherheit in Fahrzeugen zum Inhalt haben wird. Bis die Norm veröffentlicht wird (Zieldatum ist der 1. Oktober 2019 [56]) können für die Sicherheitsanalyse von Fahrzeugsystemen Verfahren eingesetzt werden, welche für IT-Systeme im Allgemeinen entwickelt wurden. Daneben existiert die US-Amerikanische SAE J3061 Norm für Informationssicherheit im Automobilumfeld, welche im Januar 2016 veröffentlicht wurde. Dieser Standard versteht sich als ein Ratgeber für die Anwendung empfohlener Verfahren zur Wahrung der Informationssicherheit im Automobil. Dabei ist die Gesamtmethodik stark an die ISO-26262 Methodik angelehnt und es sind eine Reihe von Interaktionspunkten beider Verfahren definiert [57].

4.1 Verfahren zur Gefahrenanalyse und Risikoeinschätzung

Eine der in SAE J3061 Norm vorgeschlagenen Verfahren zur Gefahrenanalyse und Risikoeinschätzung (engl. Threat Analysis and Risk Assesment, TARA) ist die *HEA-*

VENS-Methodik, die als Teil des *HEAVENS*-Projekt (engl. Healing Vulnerabilities to Enhence Software Security and Safety) entstanden ist [58]. Das Verfahren verbindet Microsofts *STRIDE*-Methode zur Gefahrenanalyse von Softwaresystemen [59] und die *EVITA*-Methodik (entwickelt im Rahmen des europäischen EVITA-Projekts [60][61]) zur Risikoeinschätzung.

4.1.1 HEAVENS Methodik

Die HEAVENS-Methodik [27] folgt einem typischen Ablauf für eine Top-Down Sicherheitsanalyse: Identifizierung der essentiellen Werte (engl. Assets) des Systems, Gefährdungsanalyse und anschließende Risikoeinschätzung. Zuletzt können entsprechend der gefundenen Risiken die notwendigen Sicherheitsanforderungen an das System aufgestellt werden. Abbildung 4.1 zeigt den Ablauf der HEAVENS-Methodik, unterteilt in die vier Einzelschritte und deren jeweilige Ausgabe.

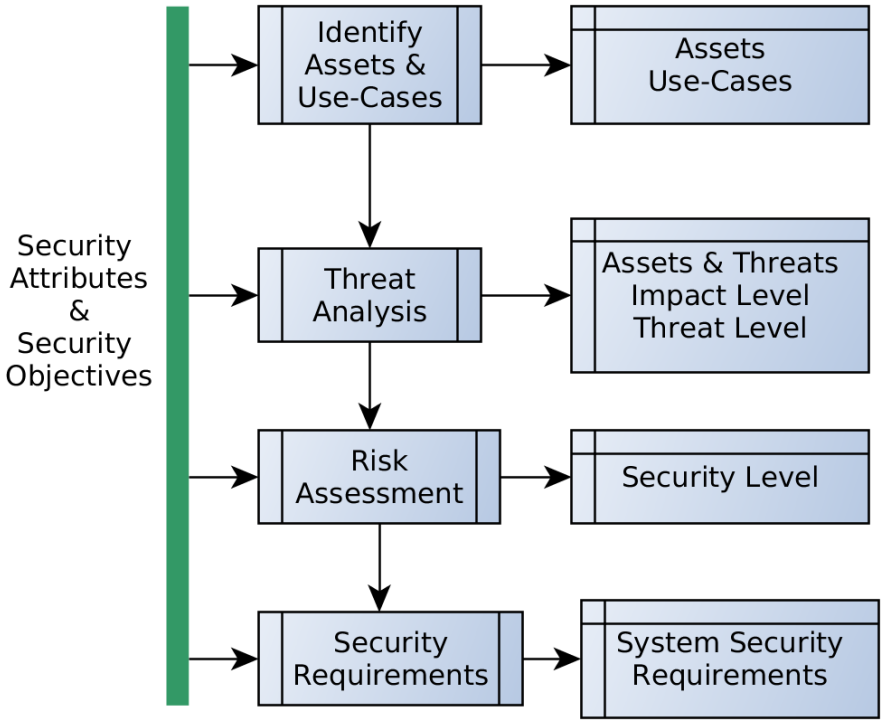


Abbildung 4.1: Ablauf der HEAVENS-Methodik zur Gefahrenanalyse und Risikoeinschätzung.

Das HEAVENS-Verfahren stützt sich dabei auf die sog. *Security Attributes* und *Security Objectives*. Abbildung 4.2 zeigt den Zusammenhang zwischen Sicherheitseigenschaften und Schutzzielen. *Security Attributes* bezeichnet die Sicherheitseigenschaften (siehe Kapitel 2.3.1) des Systems. *Security Objectives* bezeichnen dagegen die Schutzziele verschiedener

Interessensgruppen. Die HEAVENS-Methode betrachtet die folgenden Schutzziele, wobei die Punkte 4a und 4b aufgrund häufiger Überschneidung gemeinsam betrachtet werden:

1. Safety - Betriebssicherheit des Systems und Schutz der Gesundheit von Fahrer, Passagieren und anderen Verkehrsteilnehmern.
2. Operational - Aufrechterhaltung der beabsichtigten Funktionalität des Systems in seinem gesamten Umfang.
3. Financial - Verhinderung von Diebstahl oder anderen finanziell motivierten Eingriffen in das System (z. B. Veränderung des Kilometerstands zur Erhöhung des Verkaufspreises).

4a Privacy - Datenschutz und Anonymität des Fahrers sicherstellen und das geistige Eigentum der Hersteller schützen.

4b Legislation - Die Einhaltung gesetzlicher Vorschriften umsetzen, z. B. Manipulationen zur Steigerung der Motorleistung ohne Abnahme durch den TÜV verhindern.

Für die Gefährdungsanalyse verwendet die HEAVENS-Methodik die STRIDE-Gefährdungsklassen, welche auf die Sicherheitseigenschaften des Systems abgebildet werden. Für die Risikoeinschätzung wird dagegen die EVITA-Methodik eingesetzt, auf deren Basis ein zusammenfassendes *Sicherheitslevel* für jedes untersuchte Gefährdungs-Asset-Paar bestimmt wird.

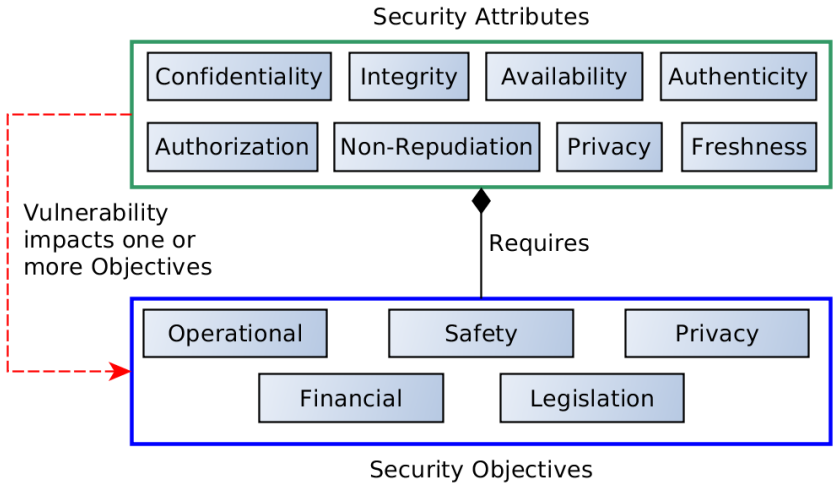


Abbildung 4.2: Zusammenhang der HEAVENS Sicherheitsattribute und Sicherheitsziele.

4.1.1.1 Gefährdungsanalyse mit STRIDE

Das STRIDE-Verfahren ist ein Angreifer und Asset zentriertes Verfahren und konzentriert sich auf die Ziele eines möglichen Angreifers. Es dient vornehmlich zur Identifizierung und Kategorisierung der Gefährdungen anhand der gegebenen Systemarchitektur, verfügbarer Schnittstellen und Assets des Systems. Ein weiterer Punkt der STRIDE-Mechanik ist die Definition von sogenannte *Trust Boundaries*, Bereiche des Systems innerhalb derer keine Angriffe ausgeführt werden können. *STRIDE* ist ein Akronym für die sechs betrachteten Angriffsziele des Verfahrens:

S - Spoofing. Ein Angreifer gibt sich für einen legitimen Teilnehmer oder Nutzer aus, z. B. indem die Senderaddresse einer Nachricht verändert wird. Ein Spoofing-Angriff verletzt die *Authentication* und *Freshness* Sicherheitseigenschaften.

T - Tampering. Veränderung von gespeicherten Daten oder des Inhalts einer legitimen Nachricht durch den Angreifer. Dies betrifft die Integritätseigenschaft des Systems.

R - Repudiation. Aktionen des Angreifers können nicht zurückverfolgt werden, z. B. weil dieser sich mit den Anmeldedaten eines Nutzers an ein System anmeldet und Aktionen ausführt. Sowohl *Freshness* als auch *Nonrepudiation* sind von einer solchen Attacke betroffen.

I - Information Disclosure. Der Angreifer kann gespeicherte oder gesendete Daten auslesen - dies betrifft *Confidentiality* und *Privacy* des Systems.

D - Denial of Service (DoS). Der Angriff unterbricht die normale Funktionsweise des Systems, z. B. durch das Blockieren des Kommunikationskanals. DoS-Angriffe zielen auf die *Availability*-Eigenschaft des Systems.

E - Elevation of Privilege. Dem Angreifer gelingt es Aktionen durchzuführen, zu denen er nicht berechtigt ist, z. B. in dem eine Passwortabfrage umgangen wird. Dies stellt eine Verletzung des *Authorization*-Attributs dar.

4.1.1.2 EVITA und HEAVENS Risikoeinschätzung

Die Risikoeinschätzung der HEAVENS-Methodik bestimmt für jedes Paar aus Asset und Threat ein Sicherheitslevel. Dieses ist als Anlehnung an das *ASIL*-Level (engl. Automotive Safety Integrity Level) zu verstehen, welches in ISO-26262 definiert ist. Je höher das ASIL-Level einer Komponente, desto höher ist deren Einfluss auf die Betriebssicherheit des Gesamtsystems. Analog bedeutet ein hohes Sicherheitslevel eine möglicherweise signifikante Schwachstelle mit erheblichem Schadenspotential sollte diese von einem Angreifer ausgenutzt werden.
Zur Bestimmung des Sicherheitslevels werden in der HEAVENS-Methode zwei Kenngrößen eingesetzt: *Threat Level* und *Impact Level*. Der *Threat Level* beschreibt die Komplexität

des betrachteten Angriffs. Tabelle 4.1 zeigt die Parameter, welche zur Bestimmung betrachtet werden. Die betrachteten Parameter beschreiben das benötigtes Fach- und Insiderwissen, die Notwendigkeit für spezialisierte Werkzeuge und die Art des Zugriffs auf das System, die für einen erfolgreichen Angriff notwendig sind. Die einzelnen Werte werden aufaddiert um das *Threat Level* des Angriffs zu bestimmen. Je höher dieser Wert, desto wahrscheinlicher - da einfacher durchzuführen - ist ein Angriff.

Komponente	Einstufung	Wert
Fachwissen	Amateur	3
	Fortgeschritten (z. B. Werkstatt, Mechaniker)	2
	Experte, Hacker	1
	Expertenteam	0
Verfügbarkeit von Informationen über das System	Öffentlich	3
	Eingeschränkt (z. B. Reparaturanleitung, Nutzerhandbuch)	2
	Interna (z. B. Hersteller, Zulieferer, Kooperativen)	1
	Geheim	0
Werkzeuge	Standard	3
	Spezialisiert (z. B. herstellerspezifisch aber frei verfügbar)	2
	Maßgeschneidert	1
	Mehrere maßgeschneidert	0
Zugriff	Unlimitierter Fernzugriff	3
	Zeitweiser Fernzugriff	2
	Einfacher physischer Zugriff oder geschützter Fernzugriff	1
	Physischer Zugriff notwendig	0

Tabelle 4.1: Der HEAVENS *Threat Level* Kenngröße ist die Summe der vier betrachteten Parameter: Fachwissen, Insiderwissen, Werkzeuge und Zugriff. Je höher der Wert, desto Wahrscheinlicher ist ein Angriff auf das betrachtete System.

Der *Impact Level* gibt dagegen die Schwere der Auswirkungen des jeweiligen Angriffs auf die *Security Objectives* des Systems an. Tabelle 4.2 zeigt die Einstufung von bestimmten Einflüssen zu *Impact Level*. Die Gewichtung der einzelnen Werte zur Ermittlung des finalen *Impact Level* kann je nach Zielsetzung angepasst werden. Im Folgenden wird das Maximum aller betrachteten Parameter als *Impact Level* gewählt.

Impact Level	Betriebssicherheit	Datenschutz	Finanziell	Funktionalitäts-einschränkung
0	Keine	Keine	Keine	Keine
1	Leichte bis mittlere Verletzungen	Anonyme Daten	Gering	Geringe
2	schwere Verletzungen oder mehrere mittlere Verletzungen	Identifikation des Fahrers oder des Fahrzeugs möglich	Mittel	Mittel
3	Lebensgefährlich	Bewegungsdaten von Fahrer und Fahrzeug	Groß	Schwer
4	Lebensgefährlich für mehrere Personen	Bewegungsdaten mehrerer Fahrer	Groß, mehrere Fahrzeuge	Schwer, mehrere Fahrzeuge

Tabelle 4.2: Der HEAVENS Impact Level ergibt sich anhand des Einfluss auf die vier *Security Objectives* des Systems [57].

Die Risikoeinstufung eines Angriffs wird durch das HEAVENS-Sicherheitslevel bestimmt. Dieses unterscheidet das Risiko von betrachteten Angriffen in die folgenden, nach steigender Priorität geordneten, Klassen: QM (Qualitätsmanagement, Gegenmaßnahmen nicht zwingend erforderlich), GERING, MITTEL, HOCH und KRITISCH. Das Sicherheitslevel wird mittels der Risikomatrix (vgl. Tabelle 4.3) ermittelt, welche die *Impact Level* und *Threat Level* Parameter des Angriffs abwiegt.

	Threat Level				
Impact Level	0-2	3-5	6-8	9-11	11+
0	QM	QM	QM	QM	QM
1	QM	GERING	GERING	GERING	MITTEL
2	QM	GERING	MITTEL	MITTEL	HOCH
3	QM	GERING	MITTEL	HOCH	HOCH
4	GERING	MITTEL	HOCH	HOCH	KRITISCH

Tabelle 4.3: Die HEAVENS Risikomatrix bestimmt das Sicherheitslevel eines Angriffs anhand dessen *Impact Level* und *Threat Level*. [57] [27].

4.2 SOME/IP Analyse

Im Folgenden Kapitel wird das in Abschnitt 4.1.1 beschriebene *HEAVENS*-Verfahren zur Gefährdungsanalyse und Risikoeinschätzung auf das SOME/IP Protokoll im Umfeld mit Hinblick auf die in 2.1 beschriebenen Architektur angewendet.

4.2.1 Angreifermodell

Für die Durchführung der Sicherheitsanalyse muss zunächst ein Angreifermodell beschrieben werden. Die HEAVENS-Methode schätzt während des Risikoanalyse die von einem Angreifer benötigten Fähigkeiten, Werkzeuge, Fachkenntnisse und Zugriffsmöglichkeiten ab, weshalb das Angreifermodell abstrakt bleiben kann und lediglich die grundlegenden Annahmen zu den Fähigkeiten des Angreifers beschreiben muss. Für eine Sammlung realistischer Angriffsszenarien auf Fahrzeugsysteme kann die Arbeit von Ponikwar et al. herangezogen werden, welche 15 verschiedene Szenarien und ein zugeordnetes Angreifermodell beschreibt [62]. Die betrachteten Fälle zielen dabei vorwiegend auf die C2X-Kommunikation und weniger auf die Manipulation des Onboard-Netzwerkes ab. Im Folgenden werden deshalb drei abstrakte Angriffsszenarien vorgestellt, welche dem Angreifer verschiedene Möglichkeiten und Einschränkungen auferlegen. Zudem wird für jedes Szenario ein Sicherheitsziel definiert, welches aus Sicht der Systemsicherheit für die jeweilige Situation erreicht werden.

Applikationskontrolle Der Angreifer hat die Kontrolle über eine Softwarekomponente erlangt. Aufgrund der Komplexität des Gesamtsystems kann eine Sicherheitslücke in einer einzelnen Softwarekomponente nicht ausgeschlossen werden - aufgrund der hohen Anzahl verwendeter Komponenten ist die Existenz einer solchen Lücke sogar sehr wahrscheinlich. Der Angreifer ist in der Lage alle Aktionen durchzuführen, welche das zugrundeliegende Betriebssystem der übernommenen Applikation erlaubt. Insbesondere schließt dies Übertragungen von Nachrichten über die bereitgestellten Schnittstellen (z. B. SOME/IP) ein.
Insgesamt ist dieses Szenario das Wahrscheinlichste und kann als Ausgangspunkt für die beiden schwerwiegenderen Szenarien dienen, falls die übernommene Applikation low-level Netzwerkzugriff oder eine hohe Berechtigungsstufe besitzt. Zentrales Sicherheitsziel des Systems muss es sein, einen mehrstufigen Angriff (vgl. Jeep-Hack Beispiel in Abschnitt 2.2.3) mithilfe einer übernommenen Anwendung zu verhindern und damit die Auswirkungen einer einzelnen Sicherheitslücke auf die betroffene Applikation zu reduzieren.

Netzwerkkontrolle Der Angreifer besitzt vollständigen (Fern-)Zugriff auf das verwendete Netzwerk (*Dolev-Yao Modell*), d. h. ungesichert übertragene Nachrichten können vom Angreifer gelesen und beliebig verändert werden. Zudem kann er aufgezeichnete Nachrichten erneut senden (engl. Replay) oder vollständig neu erstellen und übertragen. Der Angreifer versucht in diesem Szenario das Gesamtsystem zu manipulieren, indem er selbst erstellte Nachrichten direkt in das Netzwerk eingespielt. Da die so erstellten Nachrichten dabei nicht über die gegebenen Kommunikationsschnittstellen der *adaptive AUTOSAR* Architektur übertragen werden, greifen sämtliche Einschränkungen des zugrundeliegenden Betriebssystems und Kommunikationsschnittstellen nicht. Dies bedeutet jedoch auch, dass sämtliche Sitzungsinformationen und eventuelle vorhandene kryptographische Schlüssel die-

ser Schichten dem Angreifer unbekannt sind, solange diese nicht unverschlüsselt übertragen wurden, oder der Angreifer auf einem anderen Weg Zugriff auf diese Informationen erlangt.
Das Szenario stellt eine stark abstrahierte Situation dar, welche eine Vielzahl von realen Angriffen abdecken soll und wird als das Standardszenario angenommen, wobei die Fähigkeiten des Angreifers überschätzt werden. Eine solche Kontrolle über das interne Netzwerk stellt realistisch betrachtet eine nicht zu vernachlässigende Hürde für potentielle Angreifer dar, wird jedoch zumeist als Standardannahme für die Analyse von Netzwerkprotokollen gewählt. Lediglich mittels direktem (z. B. über ein Diagnoseport) oder indirektem physischen Zugang zum Netzwerk (z. B. mithilfe eines manipulierten Diagnosegeräts oder Kontrolle über verbaute Switches bzw. Hubs) ist dieser Zugriff trivial. Für einen Fernzugriff muss eine Sicherheitslücke in einer externen Schnittstelle und zusätzlich in einer Applikation existieren, welche über low-level Netzwerkzugriff verfügt. Der Angreifer muss die Applikation über die externe Schnittstelle übernehmen, um über deren low-level Zugriff auf das Onboard-Netzwerk manipulierte Nachrichten übertragen zu können bzw. den Netzwerkverkehr zu lesen. Alternativ kann es dem Angreifer gelingen eine manipulierte Applikation mit diesen Fähigkeiten auf dem System zu installieren.

ECU-Kontrolle In diesem Szenario besitzt ein Angreifer vollständige Kontrolle über eine ECU, insbesondere bedeutet dies, dass der Angreifer auf alle Daten (inklusive Schlüsselinformationen) der ECU lesend und schreibend zugreifen kann. Dieses schwerwiegende Szenario stellt eine Erweiterung der einfachen Applikationskontrolle dar. Falls das zugrundeliegende System keine zusätzlichen Sicherheitsmaßnahmen vorsieht (vgl. Kapitel 5.1) sind beide Szenarien identisch. Da selbst einfache Betriebssysteme zumindest rudimentäre Sicherheitsmaßnahmen treffen (schreibgeschützte Dateien und reservierte Speicherbereiche), kann dieses Szenario jedoch nur dann realistisch eintreten, falls es dem Angreifer gelingt eine Applikation mit Administratorrechten (engl. root-access) zu übernehmen. Eine andere Möglichkeit stellen Schwachstelle in der eingesetzten Virtualisierungssoftware dar, welche durch Applikationen ausgenutzt werden können, um das zugrundeliegende Betriebssystem zu übernehmen [63]. Diese sogenannten *Guest-to-Host* Angriffe sind aufgrund der hohen Berechtigungsstufe von Virtualisierungssoftware verheerend. In Kombination mit Netzwerkkontrolle genügt ein Zugriff auf gegebenenfalls eingesetzte Schlüsselinformationen des Systems, um die Fähigkeiten des Angreifers erheblich zu steigern. Insgesamt stellt dies ein *worst-case* Szenario dar, ist zugleich aber weniger wahrscheinlich als die vorherigen Situationen. Offensichtlich kann die Sicherheit aller auf der betroffenen ECU ausgeführten Services in diesem Szenario nicht mehr sichergestellt werden. Wünschenswertes Ziel ist jedoch die Sicherheit von nicht-betroffenen ECUs und deren Kommunikation untereinander auch in dieser Situation garantieren zu können.

Zuletzt kann ein Angreifer auch vollen invasiven, physischen Zugriff auf ein Fahrzeug erlangen, in der Hoffnung zusätzliche Informationen für Angriffe auf andere Fahrzeuge

desselben Typs oder Herstellers zu erlangen. Eine Verteidigung gegen diesen letzten Punkt ist praktisch unmöglich, deshalb muss beim Entwurf der Architektur darauf geachtet werden, dass insbesondere das verwendete Schlüsselmaterial einzigartig ist und nicht für Angriffe auf andere Systeme eingesetzt werden kann (vgl. Abschnitt 2.3.2.2).

4.2.2 Schritt 1: Asset Identifikation

Die Anwendungsfälle (engl. *Use-Cases*) des SOME/IP Protokolls lassen sich direkt aus der Protokollbeschreibung aus Kapitel 3 ableiten:

1. Aufruf von Methoden anderer Services unabhängig von deren Lokalität und Empfang eventueller Rückgabewerte (*RPCs*).
2. Events anderer Services abonnieren (*Event Subscription*).
3. Felder anderer Services auslesen und verändern (*Fields*).
4. Funktionalität des eigenen Service anderen zur Verfügung stellen (*OfferService*).
5. Kenntnisse über Services erhalten, welche aktuell innerhalb des Gesamtsystems bereitgestellt werden (*FindService*).

Für die Bereitstellung dieser Anwendungsfälle werden zudem die folgenden Funktionalitäten benötigt:

6. Die *Manifest* Datei (vgl. 3.5.3). Diese dient zur Definition eines Service, erlaubt die ECU-lokale Service Discovery und enthält alle benötigten Konfigurationsdaten zum korrekten Betrieb des jeweiligen Service.
7. Service Discovery Protokoll. Für die korrekte Funktionsweise der Middleware ist die unbeeinflusste Durchführung des SOME/IP-SD Protokolls essentiell, damit benötigte Services entdeckt und genutzt werden können.
8. Systematische und systemweit eindeutige Vergabe von SOME/IP Service-ID und Service-Instance-ID. SOME/IP fordert, dass verschiedene Services unterschiedliche Service-IDs besitzen, da diese die Grundlage für die Zuordnung der SOME/IP-Kommunikation darstellen (vgl. Abschnitt 3.2).
9. Unicast-Kommunikation. Use-Cases 1, 2 und 3 erfordern direkte Client-Server Kommunikation. SOME/IP nutzt TCP/UDP-IP zur Übertragung von Unicast-Nachrichten.
10. Multicast-Kommunikation. Zur effizienten Implementierung von Event-Benachrichtigungen und des SOME/IP-SD Protokolls ist der Einsatz von Multicast-Kommunikation notwendig. SOME/IP nutzt IP-Multicast-Gruppen für diesen Zweck.

4.2.3 Schritt 2: Threat Analysis

Im Folgenden wird die STRIDE-Methodik auf die im vorherigen Schritt (4.2.2) erstellte Asset-Liste angewandt, um mögliche Angriffe auf das SOME/IP-Protokoll zu entdecken.

ID	Asset	Typ	Beschreibung
1.1	Manifest Datei	T	Veränderung der Service-Beschreibung, z. B. durch vorgetäuschtes Update oder direkten Speicherzugriff
1.2	Manifest Datei	D	Löschen des Manifest macht die Ausführung des Service unmöglich
2.1	Service-ID	T	Veränderung der Service-ID durch Änderung der Manifest-Datei
3.1	Service Discovery	S+E	Angreifer sendet unautorisierte *offerService*-Nachrichten (z. B. um eine manipulierte Service-Instanz anzubieten).
3.2	Service Discovery	T	Angreifer manipuliert SOME/IP-SD Nachrichten (z. B. um IP-Konfiguration im Optionen-Feld auf eigene Adresse zu ändern)
3.3	Service Discovery	I	Angreifer liest SOME/IP-SD Nachrichten mit oder initiiert Service Discovery durch Senden von *findService*-Nachrichten.
3.4	Service Discovery	R+D	Angreifer wiederholt eine zuvor gespeicherte SOME/IP-SD Nachrichten (z. B. zur Unterbrechung eines Service mittels Replay einer *stopOfferService*-Nachricht).
4.1	RPC	I	Mitlesen von Methodenaufruf durch Angreifer.
4.2	RPC	R+S	Unautorisierter Methodenaufruf durch Angreifer mittels aufgezeichneter SOME/IP RPC-Nachricht.
4.3	RPC	S+E	Unautorisierter Methodenaufruf durch Angreifer mittels manipulierter SOME/IP RPC-Nachrichten.
4.4	RPC	I	Auslesen der RPC-Antwortdaten.
4.5	RPC	R+S	Replay von RPC-Antwortdaten einer früheren Anfrage auf neue Anfrage (vor der eigentlichen Antwort).
4.6	RPC	S+T	Manipulation der RPC-Antwortdaten.
5.1	Events	I	Auslesen der Eventdaten.
5.2	Events	E	Unautorisiertes Abonnieren eines Events.
5.3	Events	S+D	Abonnenten-Status eines anderen Service ändern.
5.4	Events	R+S	Replay früherer Events.
5.5	Events	S+T	Modifikation der Event-Daten.

Tabelle 4.4: Threat-Asset-Liste: Angriffe auf das SOME/IP-Protokoll und die Manifest-Datei von SOME/IP-Services.

4.2.3.1 Zusammenfassung der Angriffe

Auffallend - und wenig überraschend - ist die Gemeinsamkeit vieler Angriffe. Das SOME/IP-Protokoll besitzt keinerlei Absicherung und ist deshalb für alle klassischen Angriffe eines Angreifers mit Netzwerkkontrolle (Spoofing, Tampering, Replay) anfällig. Da alle Daten unverschlüsselt übertragen werden, ist zudem eine Verletzung der Geheimhaltungseigenschaft für alle Nachrichten gegeben. Die Angriffe 4.3 und 5.2 sind dabei auch einem schwächeren Angreifer im Szenario der Applikationskontrolle möglich, soweit das zugrundeliegende System keine speziellen Vorkehrungen trifft.
Für einen Angreife mit ECU-Kontrolle kommen noch zusätzliche Angriffe auf die Konfigurations- und Anwendungsdatendaten hinzu.

4.2.4 Schritt 3: Risk Assessment

Die folgenden Tabelle (4.5) zeigt das Ergebnis der Risikoanalyse aller betrachteten Asset-Threat-Paare. Eine Begründung für die Einstufung der jeweiligen *Impact Level* und *Threat Level* Parameter folgt in den weiteren Unterabschnitten dieses Absatzes - getrennt nach den betrachteten Assets. Das Sicherheitslevel ergibt sich direkt aus den bestimmten Werten für Impact Level und Threat Level auf die HEAVENS-Risikomatrix (siehe. 4.3).

ID	Typ	Betriebssicherheit	Datenschutz	Finanziell	Funktionalität	Impact Level	Fachwissen	Systemkenntnisse	Werkzeuge	Zugriff	Threat Level	Sicherheits-level
1.1	T	4	4	4	3	4	2	2	2	3	9	KRITISCH
1.2	D	2	0	2	2	2	3	2	2	3	11	MITTEL
2.1	T	3	3	3	3	3	3	2	2	1	8	MITTEL
3.1	S	4	3	4	4	4	1	3	3	3	10	HOCH
3.2	T	2	2	2	2	2	3	3	3	3	12	HOCH
3.3	I	0	1	0	0	1	3	3	3	3	12	MITTEL
3.4	R+D	3	0	3	3	3	3	3	3	3	12	HOCH
4.1	I	0	1	0	0	1	3	3	3	3	12	MITTEL
4.2	R	4	1	4	4	4	3	3	3	3	12	KRITISCH
4.3	T	4	1	4	4	4	2	2	3	3	10	HOCH
4.4	I	0	3	3	0	3	3	3	3	3	12	HOCH
4.5	R	4	3	4	4	4	3	2	3	3	11	KRITISCH
4.6	T	4	3	4	4	4	2	2	3	3	10	HOCH

5.1	I	0	3	3	0	3	3	3	3	3	12	HOCH
5.2	E	0	3	3	0	3	2	3	3	3	11	HOCH
5.3	D+S	3	0	3	3	3	2	3	3	3	11	HOCH
5.4	R	4	0	4	4	4	3	2	3	3	11	HOCH
5.5	T	4	0	4	4	4	2	2	3	3	10	HOCH

Tabelle 4.5: Risikoabschätzung für Angriffe auf die Manifest-Datei. Vergleiche Tabelle 4.4 für eine genaue Beschreibung der einzelnen Threats.

4.2.4.1 Manifest-Datei

Für die Manipulation der Datei (1.1), benötigt ein Angreifer Zugang und Kenntnisse über den Update (Flashing) Prozess des anzugreifenden Steuergerätes. Dies kann typischerweise mithilfe standardisierter Diagnoseprotokolle und frei verkäuflicher Diagnosegeräte bewerkstelligt werden. Moderne Fahrzeuge unterstützen zudem zunehmend OTA-Updateverfahren über Fernkommunikation, welche ebenfalls für das Einspielen manipulierter Programm- und Konfigurationsdaten verwendet werden kann. Ist das Manifest nicht in einem speziell gesicherten Speicherbereich abgelegt, so ist auch die Veränderung durch andere (eventuell vom Angreifer kontrollierte) Applikationen möglich (vgl. Applikationskontrolle in Abschnitt 4.2.1). Wird ein geschützter Speicherbereich verwendet, so sinken die Zugriff- und Fachwissenparameter deutlich und damit das Sicherheitslevel auf HOCH, da der Angreifer dann entweder physischer Zugriff oder ECU-Kontrolle benötigt.
Die Kenntnisse zur Durchführung des Angriffs (Erstellung einer entsprechenden Manifest-Datei und veränderten Applikation) sind fordernd und benötigen Systemkenntnisse. Für das bloße Löschen vorhandener Daten (1.2) werden dagegen (neben dem Schreibzugriff auf die entsprechende Datei) keine besonderen Fähigkeiten seitens des Angreifers benötigt.

Die Manipulation der Manifest-Datei (und der Applikation, welche den Service bereitstellt) auf einer ECU, ermöglicht dem Angreifer beliebige Services anzubieten, auszulesen und Funktionen anderer Services zu aktivieren. Ein Angreifer kann dies nutzen, um die Rechte einer übernommenen Anwendung zu erhöhen, z. B. um einen mehrstufigen Angriff auszuführen. Im schlimmsten Fall kann der Angreifer dadurch die Steuerung sicherheitskritischer Funktionen erlangen, z. B. das Aktivieren eines Bremsassistenzsystems bei hohen Geschwindigkeiten. Der Grund für die höchste Safety, Privacy und Financial Einstufung ist die Möglichkeit zur Ausweitung des Angriffs auf andere Fahrzeuge mittels der Interfahrzeugkommunikation (Car2X) des betroffenen Fahrzeugs. Das bloße Ausschalten eines Service (1.2) hat in der Regel keine derart weitreichenden Auswirkungen, da ein reiner Funktionsausfall frühzeitig vom System erkannt und der Fahrer informiert werden kann.

4.2.4.2 Service-ID

Die Veränderung der SOME/IP Service-ID direkt im Manifest ist ein möglicher Angriff entsprechend dem bereits vorgestellten Szenario 1.1. Das benötigte Fachwissen zur Durchführung ist jedoch im Vergleich erheblich geringer, da kein völlig neues Manifest und/oder Applikationsprogramm erstellt werden muss, sondern lediglich ein einzelner Zahlenwert in der bestehenden Datei geändert werden muss. Die Anforderungen an Systemkenntnis, Werkzeuge und Zugang sind identisch. Ist die Anwendung, deren ID geändert wird, ebenfalls vom Angreifer manipuliert, sind die Auswirkungen potentiell identisch zu denen in Szenario 1.1.
Handelt es sich um eine unbeeinflusste Anwendung, sind dennoch erheblich Sicherheitsrisiken denkbar. Ein Angriff könnte z. B. die Service-Instanz IDs von zwei Entfernungssensoren manipulieren, welche von einem Parkassistenzsystem genutzt werden. Durch das Vertauschen könnte der Assistent ein Hindernis an der falschen Position entdecken und ein falsches Korrekturmanöver einleiten. Ähnliche Situationen sind für alle Fahrassistenzsysteme denkbar, die auf Sensordaten mehrerer Quellen zugreifen.

4.2.4.3 Anmerkung zu Angriffen auf die SOME/IP Kommunikation

Durch die Nutzung von Standard-IT Netzwerkanalysesoftware (z. B. Wireshark[1]) kann ein Angreifer mit Netzwerkkontrolle auch ohne spezielles Fachwissen die übertragenen SOME/IP Daten auslesen, aufzeichnen und zu einem späteren Zeitpunkt erneut senden (Replay). Das einzige Hindernis stellt die *Session-ID* dar, welche SOME/IP für das Session Handlung nutzt. Da die SOME/IP Nachrichten jedoch nicht die *Integrity*-Sicherheitseigenschaft erfüllen, stellt dies kein ernstes Hindernis dar. Durch einfaches Inkrementieren der Session-ID kann eine aufgezeichnete Nachricht vom Angreifer erneut verwendet werden, ohne dass das SOME/IP Session Handling die Nachricht als Duplikat erkennt. Da die Session-ID lediglich 16-Bit lang ist, kann ein Überlauf während der Laufzeit nicht ausgeschlossen werden - dies erlaubt eine triviale Replay-Attacke sobald die Session-ID einen zuvor genutzten Werte erneut annimmt. Zusätzlich ist der Wert leicht vorhersagbar, da für jedes Client-Server-Paar die Session-ID mit Wert 0x01 initialisiert wird, sobald der erste Methodenaufruf durch Client an den Server erfolgt.
Ein erfahrener Angreifer kann darüber hinaus komplexe Frames erstellen oder empfangene Frames beliebig verändern und in das Netzwerk einspielen. Der Aufbau der Frames ist standardisiert und die benötigten Informationen öffentlich zugänglich. Zudem lassen sich die systemspezifischen Identifier für Services und Methoden durch Beobachtung des Systems im laufenden Betrieb leicht identifizieren (vgl. Threat 3.3 in 4.4).
Einen Angreifer mit bloßer Applikationskontrolle kann trotz der Einschränkungen einen Teil der betrachteten Angriffe ausführen, sollten die Schnittstellen des Systems

[1] https://www.wireshark.org/

die Einhaltung der Applikationsbeschreibung nicht streng prüfen. Insbesondere das unautorisierte abonnieren von Eventgruppen und die Nutzung von RPCs anderer Services ist in einem solchen Fall durch die manipulierte Applikation möglich. Da dem Angreifer dabei ein direkter, lesender Zugriff auf das Netzwerk fehlt, steigt jedoch die benötigten Systemkenntnisse an, weil der Angreifer die entsprechenden Identifier nicht durch einfache Beobachtung des Systems erhalten kann.

4.2.4.4 Service Discovery

Das Erstellen einer neuen und validen Service Discovery Nachricht ist aufgrund der Komplexität des SOME/IP-SD Protokolls nicht trivial - deshalb fällt das benötige Fachwissen höher aus als bei einer Replay-Attacke oder dem Ändern eines Optionsfeldes in einer vorhandenen Nachricht. Im Folgenden werden die einzelnen Angriffe und deren potentielle Auswirkungen auf das SD-Protokoll kurz beschrieben:

3.1 Emulieren einer neuen Service-Instanz durch den Angreifer, indem eine valide *offerService*-Nachricht gesendet wird. Die hohe Gefahr für die Betriebssicherheit des Fahrzeugs entsteht durch mögliche Überlagerung bestehender Services. So kann ein Angreifer etwa ein Notbrems-Assistenz-Service mit derselben ID wie ein bestehender Service (auf einen anderen IP-Endpunkt) bereitstellen. Dies widerspricht der SOME/IP Anforderung nach Eindeutigkeit der SOME/IP Service-IDs. Das System kann die verschiedenen angebotenen Services nicht unterscheiden und das entstehende Systemverhalten ist undefiniert.
Ebenfalls möglich ist das Anbieten einer neuen Sensorinstanz - z. B. eines Radarsensors. Der SOME/IP Standard sieht vor, dass bei dem Vorhandensein mehrerer identischer Instanzen eine zufällige Instanz gewählt wird - um die Systemlast gleichmäßig zu verteilen. Zusätzlich könnte der Angreifer *endOfferService*-Nachrichten für bestehende Instanzen fälschen, um die eigene, gefälschte Instanz als die einzig verfügbare anzubieten. Insgesamt erlaubt diese Schwachstelle dem Angreifer den Betrieb eines beliebig manipulierten Services innerhalb des Systems - z. B. um gefälschte Sensordaten in das System einzuspeisen.
Privacy-relevante Daten könnte ein Angreifer durch das Anbieten eines Logging oder Diagnose-Service erhalten, wodurch auch dieses *Security Objective* verletzt wird.

3.2 Die Manipulation einer gesendeten SOME/IP-SD Nachricht kann, je nach Ausmaß der Manipulation, zu denselben Effekten wie 3.1 führen.
Für die Bewertung wird jedoch eine sehr einfache Manipulation angenommen, z. B. Änderung der IP-Adresse in einer *offerService*-Nachricht. Eine solche Änderung ist, mithilfe von Standard IT-Werkzeugen, auch von einem Laien durchführbar - während komplexe Änderungen unter dieselben Einschränkungen wie 3.1 fallen. Bereits kleine Änderungen können zu schweren Änderungen des Systemverhaltens führen, z. B. können Kollisionen von Service-Instanzen erzeugt werden oder sensible

Daten an vom Angreifer einsehbare Services umgeleitet werden. Das SOME/IP Protokoll fordert, dass verschiedene Instanzen desselben Service auf derselben ECU auf unterschiedliche IP-Endpunkte antworten müssen. Ist dies nicht gegeben kann die SOME/IP Middleware eintreffende Kommunikation nicht korrekt zuordnen und beide Instanzen erhalten alle eingehende Nachrichten.
Ein weiterer Punkt betrifft die Manipulation eventueller Sicherheitsvereinbarungen, welche während der Service Discovery getroffen werden. Ein Angreifer könnte die Optionsfelder entfernen, welche die Aushandlung einer sicheren Verbindung ermöglichen, um so eine unverschlüsselte Verbindung zu erzwingen und in der Folge weitere Angriffe zu ermöglichen.

3.3 Das Mitlesen der SOME/IP-SD Kommunikation hat an sich keine schweren Auswirkungen auf die Sicherheitseigenschaften des Systems, erlaubt dem Angreifer jedoch mehr über das System zu lernen (z. B. die im System vorhandenen Services und deren IDs), um mit diesen Kenntnissen nachfolgende Angriffe vorzubereiten. Lediglich anonyme Daten können im Rahmen des Angriffs entdeckt werden (z. B. welche Services das System anbietet). Sind solche Informationen wettbewerbsrelevant für den Hersteller, steigt die Einstufung der Privacy-Verletzung an und folglich das Sicherheitslevel.

3.4 Replay-Attacken auf die Service-Discovery ermöglicht zum einen triviale DoS-Attacken, indem ECUs mit *findOffer*-Anfragen überfordert werden und zum anderen ist das gezielte Deaktivieren von Services möglich, indem aufgezeichnete *endOfferService*-Nachrichten zu einem beliebigen Zeitpunkt gesendet werden. Die unvermittelte Unterbrechung eines Service kann je nach Systemarchitektur und betroffenen Service zu lebensgefährlichen Situation führen (z. B. Unterbrechung des Notbremsassistenten direkt nach dessen Aktivierung oder der benötigten Sensoren).

4.2.4.5 Methodenaufrufe

Der Aufbau der SOME/IP-RPC-Nachrichten ist weniger komplex als jener der Service Discovery, entsprechend sind die Anforderungen an das Fachwissen des Angreifers für die Erzeugung neuer manipulierter Frames geringer. Zur korrekten Erzeugung von Nachrichten muss zuvor der Ablauf der SOME/IP Service Discovery betrachtet werden, um zu analysieren, welche Methoden von verfügbaren Services angeboten werden. Insbesondere müssen die entsprechenden Identifier bekannt sein (vgl. 3.2). Alternativ kann ein Angreifer diese jedoch auch einfach raten (die Identifier-Länge von 16-Bit lässt dies zu).
Im Folgenden werden die Angriffe auf Methodenaufrufe (4.1 - 4.6 in Tabelle 4.4) kurz erläutert.

4.1 Durch Mitlesen und Analyse der RPC-Anfragen über einen längeren Zeitraum kann der Angreifer wichtige Informationen über das Zielsystem sammeln und damit fehlendes Wissen aus der Service Discovery Phase ausgleichen.
An sich stellt das Mitlesen der RPC-Anfragen nicht unbedingt eine Verletzung der Sicherheitseigenschaften dar, es sei denn, dass bereits in der RPC-Anfrage schützenswerte Daten (z. B. als Parameter) übertragen werden. Sollte dies der Fall sein, steigt das Sicherheitslevel entsprechend an.

4.2 Replay-Attacken auf Methodenaufrufe (dazu werden auch *getField* und *setField* Methoden gerechnet) können je nach betroffenem Service verheerende Auswirkungen auf die Sicherheitseigenschaften des Systems haben (z. B. unautorisiertes Aktivieren des Notbremsassistenten, oder Versetzen des Fahrzeugs in den Diagnosemodus). Bereits das unerwartete Aktivieren von relativ unbedeutenden Fahrzeugfunktionen (z. B. Scheibenwischer, Sitzkontrolle, Multimedia) kann den Fahrer erschrecken oder ablenken und zu Unfällen führen.

4.3 Das Erzeugen beliebiger Methodenaufrufe (oder die Manipulation bestehender Nachrichten) kann schwerere Auswirkungen als eine reine Replay-Attacken nach sich ziehen, da der Angreifer Parameterwerte außerhalb des erwarteten Spektrums übermitteln kann, welche während des normalen Betriebs nicht vorkommen würden. Dies kann undefiniertes (und damit potentiell fatales) Systemverhalten zur Folge haben, welches das Potential hat auch auf andere Fahrzeuge überzugreifen. Zugleich ist der Angriff schwieriger durchzuführen, da der Angreifer gutes Wissen über den Aufbau des Systems (z. B. *Method Identifier* und Parameter) besitzen muss, um eine beliebige Nachricht erzeugen zu können.

4.4 Das Auslesen der RPC-Antwortdaten erlaubt, je nach Methode, das Lesen sensitiver Daten durch den Angreifer (z. B. GPS-Daten und damit das Tracken des Fahrzeugs). Dies stellt eine eklatante Verletzung der Privacy-Systemeigenschaft dar und kann potentiell erhebliche finanzielle Folgen für den Hersteller (schlechte Presse, Rückrufaktionen oder rechtliche Maßnahmen) nach sich ziehen. Zudem gelangt der Angreifer durch das Mitlesen der RPC-Antwortdaten an zusätzliche Kenntnisse über den Zustand des Systems und kann dadurch weitere Angriffe besser vorbereiten.

4.5 Durch Replay der RPC-Antwortdaten einer vorherigen RPC-Anfrage lassen sich die Ergebnisse des Methodenaufrufs entsprechend manipulieren. Das Verhalten des Systems auf mehrere RPC-Antworten ist nicht genau definiert und kann je nach Implementierung variieren.
Je nach Methode und Service kann dieser Angriff zu beliebigem Fehlverhalten des Systems führen. Der Angriff ist dabei von indirekter Natur (verglichen mit 4.2), da der Angreifer abschätzen muss, wie der Empfänger die geänderten Daten interpretiert und darauf reagiert. Dies erfordert gute Systemkenntnis des Angreifers - entweder bereits vorhanden oder durch Beobachtung erschlossen. Insgesamt

ist ein gezielter Angriff dadurch schwieriger, als ein einfacher, unautorisierter Methodenaufruf.

4.6 Gezielte Manipulation der RPC-Antwortdaten ermöglicht einem Angreifer ähnliche Wirkungen wie mit einfachen Replays zu erzielen, wobei der Angriff dabei ähnlichen Einschränkungen bezüglich dem benötigten Systemwissen unterliegt.

4.2.4.6 Events

Die folgende Liste beschreibt die Angriffe 5.1 - 5.5 aus Tabelle 4.4 auf die SOME/IP-Event Mechanik genauer.

5.1 Auslesen der Eventdaten ermöglicht - neben der möglichen Verletzung der Datenschutzeigenschaft - dem Angreifer, zusätzliches Wissen über die Systemarchitektur zu gewinnen (z. B. nach welchem Event eine bestimmte Funktion auslöst).

5.2 Das unautorisierte Abonnieren eines Events ist in den Auswirkungen potentiell identisch zu 5.1, erfordert jedoch etwas höhere Kenntnisse des Angreifers, da eine korrekte *subscribeEvent*-Nachricht an den anbietenden Service gesendet werden muss. Gegenüber 5.1 kann dieser Angriff jedoch auch von einem Angreifer mit Applikationskontrolle durchgeführt werden.

5.3 Durch das Senden einer *unsubscribeEvent*-Nachricht mit der Kennung eines anderen Service kann der Angreifer diesen vom Empfang zukünftiger Events abhalten - ohne dass der betroffene Service dies bemerkt, da *subscribe/unsubscribe*-Nachrichten per Unicast abgewickelt werden. Die Auswirkungen sind ähnlich zu 3.4, da der Angreifer beliebige Events blockieren kann, welche die Grundlage für die korrekte Funktion anderer Services darstellen. Die Durchführung ist ähnlich zu 4.3 nicht trivial, da der Angreifer eine korrekt wirkende *unsubscribeEvent*-Nachricht erstellen muss.

5.4 Durch Wiederholung aufgezeichneter Events kann allen empfangenden Services ein falscher Systemzustand vermittelt werden (z. B. ein Regensensor meldet erhebliche Regenmengen obwohl kein Regen mehr fällt). Da insbesondere Sensoren die Event-Mechanik einsetzen, kann es einem Angreifer gelingen sicherheitsrelevante Sensordaten so zu verfälschen, sodass das System gefährliche Handlungen durchführt.

5.5 Das willkürliche Verändern von Eventdaten oder Erzeugen manipulierter Events kann mindestens ebenso schwere Auswirkungen wie 5.4 nach sich ziehen. Da die Eventdaten willkürlich verändert werden können, ist der Angreifer nicht auf zuvor aufgezeichnete Werte angewiesen und kann deshalb auch Werte senden, welche (weit) außerhalb des erwarteten Wertebereichs liegen. Prüft ein Empfänger nicht sorgfältig auf solche Werte kann dies dazu führen, dass der Angreifer durch das

Übertragen solcher Events Systemfehler oder -ausfälle erzeugen kann (z. B. Absturz einer Softwarekomponente nach einer Division mit Null).

4.2.5 Gegenmaßnahmen

Das Fehlen von kryptographischer Absicherung des SOME/IP-Protokolls ermöglicht eine Vielzahl von Angriffen auf die Sicherheitseigenschaften des zugrundeliegenden Systems (vgl. Tabelle 4.5).
Um die betrachteten Angriffe zu verhindern werden eine Reihe von High-Level Maßnahmen benötigt. Tabelle 4.6 zeigt eine Liste der Gegenmaßnahmen für jeden der vorgestellten Angriffe.
Bei einer genauen Betrachtung der Tabellen lassen sich die erforderlichen Maßnahmen zu den Folgenden zusammenfassen:

- Vertrauenswürdige Plattform. Manipulation des Service Manifest, der Firmware, installierter Applikationen oder von Konfigurationsdaten muss durch Sicherheitsmaßnahmen seitens der Plattform verhindert werden. Zudem muss die Einhaltung der im Service-Manifest beschriebenen Funktionen durch ein strenges Rechtemanagement seitens der Plattform erzwungen werden.
  Diese Maßnahmen liegen außerhalb des Fokus dieser Arbeit, sollten jedoch mindestens die Folgenden umfassen: sicherer Startvorgang, signierte Programm- und Konfigurationsdaten, getrennter Speicherbereich je Applikation, gesichertes und privilegiertes Updateverfahren, sowie striktes Rechtemanagement. Dadurch lässt sich ein Angreifer mit Applikationskontrolle stark einschränken und die Folgen einer vollständig kontrollierten ECU auf das Gesamtsystem verringern.
- Unicast-Kommunikation sollte zwingend die Sicherheitseigenschaften Integrität und Freshness erfüllen. Somit können unautorisierte Methodenaufrufe und die Manipulation von übertragenen Daten verhindert werden.
- Je nach Sensibilität der übertragenen Daten ist zudem die Anwendung von Verschlüsselungsverfahren zur Wahrung der Geheimhaltungseigenschaft unabdingbar. Dies erschwert einem Angreifer zudem sein Wissen über das System auszubauen.
- Multicast-Kommunikation sollte ebenfalls Sicherheitseigenschaften Integrität und Freshness erfüllen können. Nur so kann die Manipulation des SOME/IP-SD Protokolls oder der Event-Benachrichtigungen verhindert werden.
  Die Alternative - der Verzicht auf Multicast-Kommunikation - ist für das SOME/IP-Protokoll nicht möglich, ohne die Skalierbarkeit des Verfahrens stark einzuschränken.
- Je nach Sensibilität der Eventdaten ist zudem die Verschlüsselung der übertragenen Multicast-Events notwendig.

ID	Asset	Typ	Security Level	High-Level Gegenmaßnahme
1.1	Manifest Datei	T	HOCH	Trusted Platform (abgesichertes Updateverfahren, signiertes Manifest, geschützter Speicherbereich)
1.2	Manifest Datei	D	MITTEL	Trusted Platform (geschützter Speicherbereich)
2.1	Service-ID	T	MITTEL	Trusted Platform (siehe 1.1)
3.1	Service Discovery	S+E	HOCH	eindeutige Multicast-Authentifizierung
3.2	Service Discovery	T	HOCH	eindeutige Multicast-Authentifizierung
3.3	Service Discovery	I	MITTEL	Verschlüsselung der SOME/IP-SD Nachrichten
3.4	Service Discovery	R+D	HOCH	Integritätsschutz, zusätzlicher Freshness-Wert für alle Nachrichten
4.1	RPC	I	MITTEL	Verschlüsselung der SOME/IP Methodenaufrufe
4.2	RPC	R+S	HOCH	Integritätsschutz, zusätzlicher Freshness-Wert für alle Nachrichten
4.3	RPC	S+E	HOCH	Integritätscheck für SOME/IP Methodenaufrufe
4.4	RPC	I	HOCH	Verschlüsselung der SOME/IP Antwortframes
4.5	RPC	R+S	HOCH	Integritätsschutz, zusätzlicher Freshness-Wert für alle Nachrichten
4.6	RPC	S+T	HOCH	Integritätsschutz der SOME/IP Antwortframes
5.1	Events	I	HOCH	Verschlüsselung der Eventdaten
5.2	Events	E	HOCH	Integritätscheck von SOME/IP *subscribe*-Nachrichten
5.3	Events	S+D	HOCH	Integritätscheck von SOME/IP *unsubscribe*-Nachrichten
5.4	Events	R+S	HOCH	eindeutige Multicast-Authentifizierung, zusätzlicher Freshness-Wert für alle Nachrichten
5.5	Events	S+T	HOCH	eindeutige Multicast-Authentifizierung

Tabelle 4.6: Threat-Asset Liste mit Sicherheitseinstufung und möglicher High-Level Sicherheitsmaßnahme zur Vermeidung des Angriffs.

5 Sicherheitsmechanismen für sichere SOME/IP Kommunikation

Im vorangegangenen Kapitel wurde das SOME/IP Protokoll auf mögliche informationstechnische Schwachstellen untersucht und eine Reihe von Gefahren ermittelt. Tabelle 4.6 listet diese auf und beschreibt zudem High-Level-Maßnahmen, um diesen entgegenzuwirken. Im Folgenden sollen existierend Sicherheitsprotokolle und -verfahren auf deren Tauglichkeit zur Absicherung des SOME/IP Protokolls untersucht werden. Das Kapitel teilt sich in drei Abschnitte auf. Der erste Abschnitt (5.1) beschreibt die Annahmen und Voraussetzungen, die an eine ECU für einen sicheren Betrieb zu setzen sind. In Kapitel 5.2 wird ein weit verbreiteter Ansatz zur Absicherung der internen Kommunikation in Automobilen (engl. *Secure On-Board Communication*, SecOC) betrachtet und eine mögliche Anwendung auf das betrachtete System diskutiert. Die weiteren Abschnitte (5.3 und 5.4) untersuchen bekannte Sicherheitsprotokolle auf deren Eignung zur Absicherung der verschiedenen Aspekte des Systems.

5.1 Trusted Plattform

Die ersten drei betrachteten Threats (1.1, 1.2 und 2.1) zielen auf eine Manipulation von auf der ECU gespeicherten Daten (Manifest Datei, Konfigurationsdaten, Programmcode) ab. Zusätzlich sind auch Angriffe auf die Plattform als solche denkbar (z. B. unautorisiertes Aufspielen einer manipulierten Firmware).
Die Absicherung gegen solche Maßnahmen liegt außerhalb des Fokus dieser Arbeit, weshalb im Folgenden lediglich eine Reihe von Annahmen getroffen werden, welche von den eingesetzten ECUs zugesichert werden sollten.

Sicherer Startvorgang Um eine Manipulation des Startvorgangs (z. B. Booten einer manipulierten Firmware-Version) zu verhindern, sollten Maßnahmen zur Absicherung des Startvorgangs vorgenommen werden. Ein bekanntes Prinzip für den sicheren Systemstart wird als *Chain of Trust* bezeichnet. Dazu wird nach Abschluss jeder Bootsequenz die nächste Stufe des Prozesses zunächst mithilfe kryptographischer Verfahren gegen gespeicherte digitale Signaturen authentifiziert [64].

Sicherer Speicher ECUs müssen sowohl flüchtige als auch persistente sichere Speicherbereiche zur Verfügung stellen. Zugriff auf dort gespeicherte Informationen

muss autorisierten Anwendungen (z. B. dem Betriebssystem) oder Personen vorbehalten sein. Wichtige Daten (z. B. Manifest Datei oder Schlüsselmaterial) sollten ausschließlich in sicherem Speicher abgelegt werden.

Getrennte Ausführung Das zugrundeliegende Betriebssystem sollte die Ausführung von Programmen logisch trennen, um Wechselwirkungen zu verhindern. Die Ausführung in getrennten virtuellen Adressbereichen ist ein (erster) Schritt zur Isolation von ausgeführten Programmen. Mit ARMs *TrustZone* [65], Intels *Software Guard Extension* und NXPs *Security Engine* [66] existieren verschiedene Implementierungen, um die getrennte Ausführung verschiedener Softwarekomponenten hardwareseitig zu unterstützen.
Dies ist von entscheidender Bedeutung, damit ein Angreifer mit Applikationskontrolle, nicht direkt vollständige Systemkontrolle übernehmen kann.

Rechtemanagement Das Betriebssystem sollte in der Lage sein, Verstöße von Applikationen gegen gewährte Rechte zu entdecken und verhindern. Dazu bietet sich die Erstellung von Attributzertifikaten für vertrauenswürdige Anwendungen durch den Hersteller an (vgl. Beispiel in Abschnitt 2.3.2.7). Anhand der Informationen des Service-Manifests (vgl. Abschnitt 2.1.1) können restriktive Rechte für jede Anwendung erstellt und vom Betriebssystem zur Ausführungszeit geprüft werden. Dies verhindert ein Überschreiten der Rechte durch eine installierte Anwendung und bietet sowohl ein starkes Autorisationskriterium als auch eine solide *zweite Verteidigungslinie*, um die Auswirkungen einer ausgenutzten Schwachstelle zu verringern. Ein Angreifer mit Applikationskontrolle ist durch das Rechtemanagement auf den gesteckten Rahmen der übernommenen Anwendung beschränkt. Dies gilt offensichtlich nur, solange dem Angreifer nicht zusätzlich eine Manipulation der zugrundeliegenden Berechtigungen gelingt.
Die Wirksamkeit eines solchen Ansatzes ist leicht am Beispiel des Jeep Hacks (vgl. Abschnitt 2.2.3) zu sehen. Wäre die Software der übernommenen Anwendung durch das Betriebssystem auf die notwendigen Funktionen beschränkt gewesen, so hätten die Autoren die angeschlossene ECU nicht mit einer manipulierten Firmware überschreiben können. Anstatt nahezu vollständige Kontrolle über das Fahrzeug zu übernehmen, wäre der Angriff somit auf die Funktionen der übernommenen Connectivity Software beschränkt gewesen.
Für den Rest der Arbeit wird die Annahme getroffen, dass das Betriebssystem bzw. die Middleware in der Lage ist, die Rechte eines Service zur Durchführung einer Aktion (z. B. zum Beitritt in eine Eventgruppe, oder dem Aufruf einer Methode eines anderen Service) wirksam zu prüfen. Dies kann entweder durch einfaches Prüfen des zugehörigen Manifest oder anhand eines individuellen Attributzertifikats erfolgen. Alternativ kann die Verwendung eines entsprechenden Betriebssystem erfolgen, welches ein strenges und feinmaschiges Rechtemanagement implementiert, z. B. *Security-Enhanced Linux* (SELinux). Die Open Source Kooperation OSADL eG (engl. *Open Source Automation Development Lab*), arbeitet an der Entwicklung einer Linux Variante für den Einsatz in industriellen und sicherheitskritischen Be-

reichen (u.a. im Automobilsektor) [67]. Ziel ist die Entwicklung eine zertifizierten Betriebssystem für den Einsatz in sicherheitskritischen Echtzeitsystemen.

Signierte Updates Aktualisierungen der Firmware müssen in einer sicheren, autorisierten Weise durchgeführt werden. Dies kann z. B. durch das Signieren von Updates mit einem Herstellerzertifikat erfolgen und nur Anwendungen gestattet werden, welche explizit dazu berechtigt sind (z. B. einen entsprechenden Eintrag im Attributzertifikat besitzen). Zudem sollte eine Möglichkeit bestehen, kritische Updates innerhalb kürzester Zeit zu verbreiten, damit eventuelle Sicherheitslücken zeitnah durch den Hersteller geschlossen werden können. Insbesondere bedeutet dies, dass Updates unabhängig von regulären Serviceintervallen verbreitet werden müssen. Dazu eignen sich OTA-Updateverfahren.

Durch diese Maßnahmen wird zum einen die Möglichkeiten eines Angreifers mit Applikationskontrolle stark eingeschränkt und zum anderen die Schwierigkeit für den Angreifer vollständige ECU-Kontrolle zu erreichen erheblich erhöht. Ist das Rechtemanagement in der Lage alle potentiellen Verstöße gegen die Applikationsbeschreibung zu erkennen und verhindern, so ist ein Angreifer mit Applikationskontrolle auf die Funktionen der übernommenen Applikation eingeschränkt. Ein sicheres, signiertes Updateverfahren verhindert zudem, dass ein Angreifer mit ECU-Kontrolle andere ECUs des System mit einer manipulierten Firmware-Variante überschreiben und damit seine Systemkontrolle ausweiten kann.

5.2 Secure Onboard Communication

SecOC beschreibt eine Methode zur Absicherung übertragener Daten unabhängig von der zugrundeliegenden Bus-Technologie und ist als (optionale) BSC in AUTOSAR classic (ab Version 4.2) enthalten und spezifiziert [68][69]. SecOC wird folglich in aktuellen Fahrzeuggenerationen verbreitet zur Absicherung der sicherheitskritischen fahrzeuginternen (CAN-Bus) Kommunikation eingesetzt, weshalb der Gedanke für einen Einsatz dieses Verfahrens auch für die Absicherung der SOME/IP-Kommunikation nahe liegt.

Die SecOC-Methodik setzt symmetrische MAC-Verfahren (typischerweise AES-CMAC) in Kombination mit einem *Freshness*-Wert ein, um die Sicherheitseigenschaften Integrität, Authentizität und Freshness von übertragenen Daten zu wahren. Der Freshness-Wert kann dabei als einfacher Zähler oder mithilfe von Zeitwerten (absolut oder Betriebsstunden) implementiert werden. Verbreitet ist auch die Nutzung mehrerer Quellen zur Bestimmung des Wertes.
Die Wahl rein symmetrischer Verfahren und die Tatsache, dass Varianten zum Einsatz gekürzter (engl. truncated) MACs bzw. Freshness-Werte existieren, unterstreicht eines der zentralen Ziele von SecOC: Das Erreichen der Sicherheitsziele mit möglichst geringem

Overhead in Bezug auf Rechenleistung, Speicherverbrauch und Kommunikationsoverhead.

5.2.1 Analyse

Direkt auffällig ist der systematische Verzicht auf Verschlüsselung der Nutzdaten, wodurch die Geheimhaltungseigenschaft verletzt wird. Dies erlaubt einem passiven Angreifer mit Netzwerkkontrolle sämtliche übertragene Daten zu lesen, was insbesondere im Hinblick auf datenschutzrelevante Informationen, etwa zum aktuellen Standort, kritisch zu sehen ist.
Ein weiterer problematischer Aspekt ist die Tatsache, dass der SecOC-MAC lediglich über die Nutzdaten und den Freshness Wert berechnet wird, nicht jedoch eventuelle Headerinformationen der zugrundeliegenden Kommunikationsschichten miteinbezieht. Dies kann, je nach verwendetem Protokoll, zu einer Reihe möglicher Angriffe führen, insbesondere wenn derselbe Schlüssel für die Kommunikation mit mehreren Teilnehmern eingesetzt wird. In einem solchen Fall kann ein Angreifer zum Beispiel einen validen Frame an einen anderen Empfänger umleiten. Dies kann zu gefährlichen Reaktionen des Systems führen, z. B. wenn der Informationsstrom von zwei Sensoren vertauscht wird. Der Empfänger kann dies Anhand einer einfachen SecOC-Prüfung nicht erkennen und akzeptiert die Nachrichten.
SecOC unterscheidet Unicast- und Multicast-Kommunikation nicht, und verwendet dieselben symmetrischen MAC-Verfahren in beiden Situationen. Das SecOC-Authentizitätskriterium (Kenntnis des symmetrischen Schlüssels) ist folglich für Multicast erheblich schwächer als bei einer *1:1* Kommunikation. Hat der Angreifer die Kontrolle über eine ECU (oder den symmetrischen Schlüssel) erlangt, so kann dieser sich für jedes beliebige Gruppenmitglied ausgeben, ohne dass eine SecOC-Prüfung dies erkennen kann (vgl. Diskussion zu Multicast-Sicherheitseigenschaften in Abschnitt 5.4.3).
Ein weiteres Problem für die Anwendung von SecOC für ein dynamisches System mit langer Lebenszeit, stellt die Verwaltung der Schlüssel dar. Schlüsselverwaltung (insbesondere Generierung und Austausch) ist nur rudimentär definiert - und erfordert ebenfalls ein von allen Parteien geteilten symmetrischen Schlüssel. Typischerweise werden alle Schlüssel statisch bei der Systemkonfiguration vergeben und ein Wechsel im laufenden System ist nicht vorgesehen [69]. Durch die statischen Schlüssel und die lange Lebenszeit des Systems verletzt dies offensichtlich die Anforderungen an den Umgang mit Schlüsselmaterial (siehe Kapitel 2.3.2.2). Ebenfalls ist bei Verwendung statischer Schlüssel ein dynamisches System mit wechselnden und eventuell temporären Kommunikationspartnern kaum umsetzbar.
Zuletzt ist anzumerken, dass bei der Verwendung von gekürzten MACs und Freshness-Werten das erreichte Sicherheitslevel des Verfahrens stark verringert wird. Dies kann das System, insbesondere bei stark gekürzten Werten, anfällig gegen triviale Brute-Force Angriffe machen.

5.3 Unicast-Kommunikation

Unicast-Kommunikation beschreibt die klassische Form der Datenübertragung, bei der zwei Kommunikationspartner Nachrichten über einen gemeinsamen Kanal austauschen. Mit dem Wechsel von spezialisierten Bussystemen wie dem CAN-Bus hin zu standardisierter, IP-basierter Kommunikation ist der Einsatz von klassischen IT-Sicherheitsprotokollen denkbar. Diese besitzen in der Regel den Vorteil ausgiebig erprobt und von Sicherheitsforschern auf Schwachstellen untersucht worden zu sein. Zudem stehen in der Regel diverse Implementierungen zur Verfügung, welche für verschiedene Einsatzzwecke optimiert sind und regelmäßig aktualisiert werden.

5.3.1 IP-basierte Sicherheitsprotokolle

Aufgrund der weiten Verbreitung von IP-basierter Kommunikation in IT-Netzwerken existieren eine ganze Reihe standardisierter Sicherheitsprotokolle zu deren Absicherung: TLS (engl. Transport Layer Security) [70], IPSec [71] und MACsec [72]. Jedes dieser Protokolle setzt auf einer anderen OSI-Schicht an und hat unterschiedliche Anforderungen an das zugrundeliegende IT-System.
MACsec besitzt eine zentrale Komponente (den *Kerberos*-Authentifizierungsserver) und ist deshalb nicht für das vorliegende System geeignet. Ein Ausfall des Authentifizierungsservers könnte dazu führen, dass eine ECU (z. B. nach einer Standby-Phase) von der sicheren Kommunikation ausgeschlossen wird, da keine Authentifizierung der Komponente mehr erfolgen kann.
IPSec ist ein sehr komplexes Protokoll, welches eine Vielzahl von Optionen bereitstellt. Aufgrund dieser Komplexität (welche zu einer unsicheren Konfiguration führen kann) wird das Protokoll von Sicherheitsforschern nicht empfohlen [73].
TLS stellt seit Jahren den de facto Standard für die sichere Kommunikation über das Internet dar und wurde mehrfach überarbeitet, um aufgedeckte Schwachstellen zu beheben. Der TLS Standard ist für die Absicherung von zuverlässigen, verbindungsorientierten Kommunikationskanälen entworfen und lässt sich folglich direkt auf TCP-Verbindungen anwenden. Für UDP-basierte Kommunikation existiert mit DTLS (engl. Datagram TLS) eine minimal abgewandelte Version des TLS-Standards, welche die Besonderheiten von UDP adressiert.
SOME/IP überträgt Nachrichten, je nach Ausprägung (vgl. Abschnitt 3.6), entweder per UDP oder TCP. Im Gegensatz zu UDP wird die Unterstützung von TCP vom SOME/IP Standard nicht gefordert. Die Wahl von UDP als Grundlage für die Onbaord-Kommunikation ist dabei leicht zu erklären.

Verbindungslos UDP ist ein verbindungsloses Protokoll, d. h. Server und Client müssen keine individuelle Zustandsmaschinen je Verbindung speichern, da die Datenpakete direkt übertragen werden. Zusätzlich entfällt der Drei-Wege-Handschlag, welcher benötigt wird um eine TCP-Verbindung aufzubauen.

ACK-Nachrichten Die Verlustrate von UDP-Paketen ist in kleinen, lokalen Netzwerken mit kurzen Übertragungsstrecken sehr gering. Folglich sind die ACK-Nachrichten, welche TCP verwendet um das korrekte Eintreffen eines Pakets an den Sender zu signalisieren, zumeist unnötig. Dadurch sinkt die Bandbreite des Netzwerks, welche für die Übertragung von Nutzdaten bereitsteht.

5.3.2 Transport Layer Security (TLS)

Das Ziel des TLS-Protokolls ist der Aufbau eines verschlüsselten und authentifizierten Kommunikationskanals zwischen zwei IP-Endpunkten. Der TLS-Handschlag authentifiziert beide Kommunikationsparteien anhand von digitalen Zertifikaten, führt einen Schlüsselaustausch durch und handelt die verwendeten Verfahren für Verschlüsselung und Authentifizierung der Applikationsdaten aus. Das Protokoll beschreibt dazu zwei getrennte Phasen: *Handshake* und *Record Protocol*. Letzteres beschreibt das Frame-Layout (Maximallänge, Headerinformationen) und den Ablauf des sicheren Nachrichtenaustausch, nach Abschluss des Handshake-Protokolls.
Im Folgenden wird das Handshake-Protokoll detailliert vorgestellt, da dieses für das Zustandekommen einer sicheren Verbindung entscheidend ist.

5.3.2.1 TLS-Handschlag

Der TLS-Handschlag definiert in Version 1.3 drei zentrale Nachrichten: *ClientHello*, *ServerHello* und *Client Finish* (vgl. Abbildung 5.1) [74].

ClientHello Die erste ausgetauschte Nachricht wird vom Initiator (*Client*) erzeugt und gesendet. Sie enthält die (einmalig verwendete) Zufallszahl N_C, eine Reihe von Informationen und Anfragen (z. B. Protokollversion, unterstützte Zertifikatstypen), eine geordnete Liste von unterstützten *CipherSuites* und Schlüsselaustauschinformationen $KeyExchange_C$. Wird dazu ein Diffie-Hellman Ephemeral (DHE, vgl. Perfect Forward Secrecy) Verfahren eingesetzt, kann die *perfect forward secrecy* Sicherheitseigenschaft erreicht werden.
Mit *CipherSuite* wird eine fest definierte Kombination von einem symmetrischen Verschlüsselungsverfahrens und einer kryptographisch sicheren Hashfunktion bezeichnet. Diese werden eingesetzt, um die Kommunikation zwischen Client und Server nach Abschluss des Handschlags abzusichern. In der kommenden Version 1.3 unterstützt TLS nur noch wenige (aktuell fünf) CipherSuites, wobei alle Verschlüsselungsverfahren in einem AEAD-Modus betrieben werden, d.h. übertragene Nachrichten sind sowohl verschlüsselt als auch mittels einer MAC authentifiziert.

ServerHello Nach Erhalt eines *ClientHello* antwortet der Server mit der *ServerHello*-Nachricht. Diese enthält eine (ebenfalls einmalige) Zufallszahl N_S, die gewählte Ciphersuite und den Schlüsselaustauschinformationen des Servers $KeyExchange_S$.

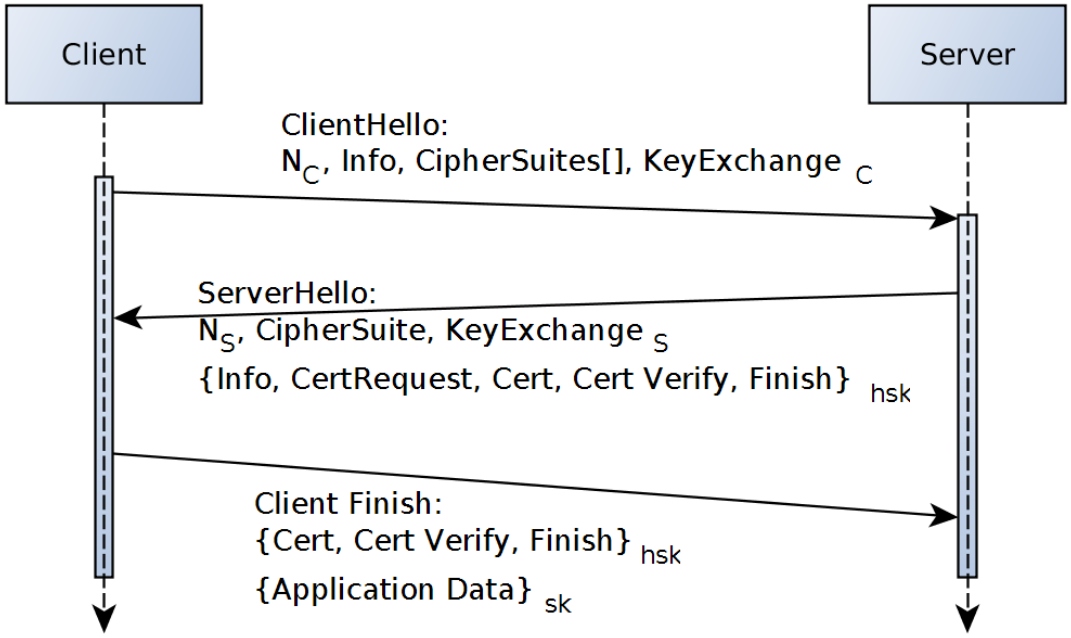


Abbildung 5.1: Sequenzdiagramm des TLS 1.3 Handshake mit zertifikatsbasierter Authentifizierung von Server und Client.

Aus $KeyExchange_C$ und $KeyExchange_S$ kann zudem der *Handshake Session Key* (hsk) erzeugt werden. Mit diesem werden alle folgenden Nachrichtensegmente des Handschlags symmetrisch verschlüsselt und authentifiziert.
Im zweiten, verschlüsselten Teil der *ServerHello*-Nachricht wird, neben zusätzlichen Informationen und der Anforderung des Client-Zertifikats, das Serverzertifikat und der Nachweis über den Besitz des privaten Schlüssels übertragen. Der Nachweis erfolgt, indem der Server das sog. *Transscript* (alle bisher ausgetauschten Nachrichten) mit seinem privaten Schlüssel signiert. Der Client kann diese Signatur mittels des übertragenen öffentlichen Schlüssels des Servers prüfen. Die letzte Komponente der Nachricht ist der sogenannte *Finish*-Block, der den Hashwert über alle bisher gesendeten Nachrichten unter Verwendung des ausgehandelten Schlüssels enthält.

ClientFinish Die letzte Nachricht des Handschlags enthält das Zertifikat des Clients und dessen Nachweis über Besitz des zugehörigen privaten Schlüssels, sowie ein erneuten *Finish-Block*. Alle Nachrichtenkomponenten dieser Nachricht sind mit dem ausgehandelten Schlüssel (hsk) verschlüsselt.
Neben diesen abschließenden Authentifizierungsinformationen, kann der Client zudem erste Applikationsdaten übertragen. Diese sind mit dem symmetrischen Sitzungsschlüssel *Session Key* (sk) verschlüsselt, welcher mithilfe einer KDF aus (hsk) erzeugt werden kann. Durch erneute Anwendung der KDF kann der Sit-

zungsschlüssel nach einer gewissen Zeit gewechselt werden, ohne einen erneuten Schlüsselaustausch durchführen zu müssen.

5.3.2.2 DTLS Änderungen

DTLS führt eine zusätzliche Phase, vor dem eigentlichen Start des TLS-Handschlags ein [75]. Nach der Übertragung der *ClientHello*-Nachricht, erzeugt der Server ein (für die Verbindung eindeutiges) Cookie und sendet dieses als Teil einer *HelloRetryRequest*-Nachricht an den Client. Der Client sendet seine ursprüngliche *ClientHello*-Nachricht erneut, fügt jedoch das empfangene Cookie hinzu. Ein solches Cookie kann z. B. durch die Berechnung des (kryptographisch sicheren) Hashwerts der *ClientHello*-Nachricht erzeugt werden. Da diese eine (einmalige) Zufallszahl enthält, ist der entsprechende Hashwert ebenfalls eindeutig.

Die weiteren Änderungen von DTLS betreffen direkt die Eigenschaften von UDP:

Sequenznummer & Epoche DTLS-Header (sowohl Handschlag als auch *Record Protocol*) enthalten eine explizite 48-Bit lange Sequenznummer und ein zusätzliches 16-Bit langes *Epoch* Feld. Jede neue gesendete Nachricht inkrementiert den Sequenznummer-Zähler. Dies ermöglicht dem Empfänger eventuell verpasste oder verlorene Pakete zu erkennen. Zudem schützt die Nutzung einer expliziten Sequenznummer vor Replay-Attacken. Bei dem *Epoch* Feld handelt es sich ebenfalls um einen Zähler, der jedoch nur bei jedem Schlüsselwechsel inkrementiert wird. Dies erlaubt einem Empfänger einen eventuell verpassten Schlüsselwechsel zu bemerken oder (innerhalb eines engen Zeitrahmens) den Einsatz des korrekten (veralteten) Schlüssels zur Verarbeitung eines verspätet empfangenen Pakets.

Timeout-Retransmit Für den Handschlag wird eine einfache Timeout-basierte Retransmit/ACK Zustandsmaschine definiert. Dies ist notwendig, um den Handschlag trotz eventueller Paketverluste oder Neuordnung der Pakete korrekt durchzuführen.

Fragmentierung Da Handschlag-Nachrichten möglicherweise länger als die maximale Framelänge (typischerweise 1500 Bytes) sind, definiert der DTLS-Standard zudem eine explizite Fragmentierung mithilfe von *fragment_offset* und *fragment_length* Feldern.

5.3.2.3 *Session Resumption* und *Early Data*

(D)TLS 1.3 bietet verschiedene Versionen des Handschlags für verringerte Latenz zwischen Kommunikationsbeginn und Übertragung der ersten Applikationsdaten. Besitzen beide Seiten einen gemeinsamen, symmetrischen Sitzungsschlüssel (z. B. aus einem vorherigen, vollständigen (D)TLS-Handschlag), so kann dieser als implizite Authentifizierungsinformation eingesetzt werden und so der Austausch und Verifikation von

Zertifikaten übersprungen werden. Dazu definiert der Standard das Erstellen eines sogenannten *NewSessionTicket* durch den Server. Der Sitzungsschlüssel wird dazu eindeutig an ein Label, eine Ciphersuite und Gültigkeitsdauer gebunden [74] und an den Client gesendet.
Der Client kann bei dem nächsten Verbindungsaufbau dieses Ticket als Teil seiner *ClientHello*-Nachricht übertragen, um die vorherige Sitzung fortzusetzen. In [76] wird zudem ein Verfahren beschrieben, um Tickets vollständig auf den Clients zu speichern, sodass der Server keine eigenen Zustandsinformationen vorhalten muss. Für das vorliegende System ist dies jedoch nicht notwendig, da die Anzahl der zu speichernder Tickets überschaubar ist.
Um die *perfect forward secrecy* Eigenschaft zu erhalten, kann ein Diffie-Hellman Schlüsselaustausch durchgeführt werden, um einen neuen Sitzungsschlüssel auszuhandeln. Der alte Sitzungsschlüssel wird lediglich zur Absicherung dieses Austausches eingesetzt, nicht jedoch für den Schutz der folgenden Kommunikation. Dies bedeutet auch, dass der Server bereits als Teil seiner *ServerHello*-Nachricht Applikationsdaten an den Client senden kann, welche mit dem neuen Sitzungsschlüssel verschlüsselt sind. Da in dieser Situationen Applikationsdaten nach genau einer *Round-Trip Time* übertragen werden, wird diese Version des Handschlag als *1-RTT* bezeichnet.
Mit (D)TLS Version 1.3 wurde zusätzlich eine *0-RTT* Version eingeführt, welche jedoch verringerte Sicherheitseigenschaften aufweist. Dazu wird der - aus einer vorherigen Verbindung - geteilte Sitzungsschlüssel vom Client direkt eingesetzt, um Applikationsdaten verschlüsselt und authentifiziert zu übertragen. Diese Daten profitieren dabei nicht von der *perfect forward secrecy* Eigenschaft und sind - ohne zusätzliche Sicherheitsvorkehrungen des Servers - nicht vor Replay-Attacken geschützt. In dieser Weise übertragene Daten werden als *Early Data* bezeichnet. Um Replay-Angriffe auf *Early Data* zu vermeiden, kann der Server eine Liste aller vergebenen Tickets verwalten. Nachdem ein Ticket zur Fortsetzung einer Verbindung eingesetzt wurde, wird es aus der Liste entfernt und somit eventuelle Replay-Attacken verhindert. Dies erfordert jedoch das Speichern aller vergebenen Tickets und verhindert die Auslagerung des Tickets auf den Client. Im vorliegenden System ist der Einsatz von *0-RTT* durchaus möglich, da die Zahl der zu verwaltenden Tickets, aufgrund der kleinen Anzahl von ECUs innerhalb des Systems, gering ausfällt. Der Verlust der *perfect forward secrecy* Eigenschaft muss jedoch sorgfältig geprüft werden. Sollten *Early Data* hochsensible Daten enthalten (z. B. Standortinformationen o.ä.), so sollte die sichere *1-RTT*-Variante zum Einsatz kommen. Werden jedoch lediglich einfache Konfigurationsinformationen (z. B. *offerService* Nachrichten als Teil des SOME/IP-SD Protokolls) als Early Data gesendet, kann der Einsatz von *0-RTT* aufgrund der verringerten Latenz durchaus gerechtfertigt sein.

5.3.2.4 Record Layer Header

Die meisten Details des (D)TLS Record Layer Protokolls sind im Folgenden nicht weiter von Bedeutung, jedoch ist die Länge und der Inhalt des Headers von Interesse für

die Anwendung auf das betrachtete System. Der TLS-Header benötigt 24 Byte für die notwendigen Headerinformationen und die symmetrische MAC der Nachricht, während die DTLS-Version - aufgrund der Nutzung einer expliziten Sequenznummer und des Epochen-Feldes - insgesamt 32 Byte benötigt (vgl. Abbildung 5.2). Die Berechnung basiert dabei auf der Annahme, dass eine 16 Byte lange MAC für die Authentifikation eingesetzt wird - dies ist der Fall für alle in TLS 1.3 spezifizierten Ciphersuites.

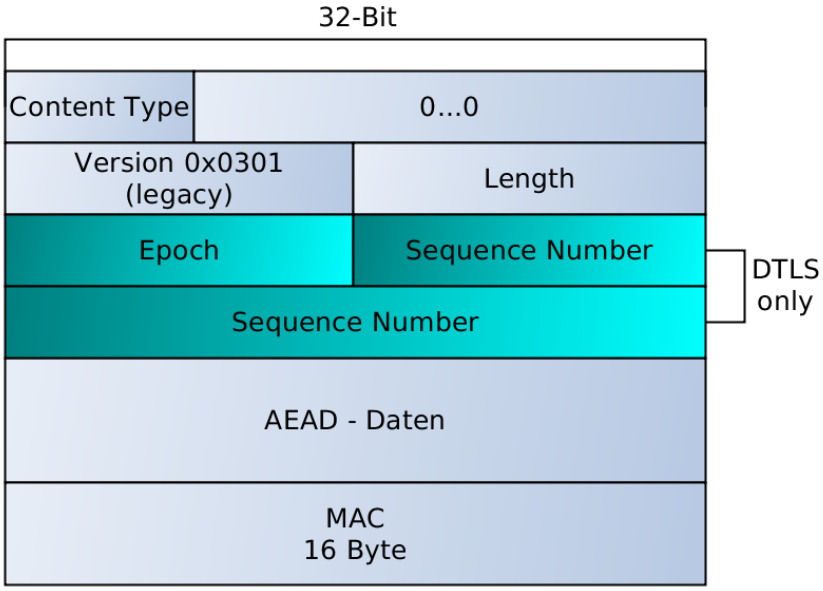


Abbildung 5.2: Aufbau eines (D)TLS Record-Layer Protokolls zur sicheren Übertragung von Daten nach erfolgtem Handschlag. Die TLS Version des Headers nimmt 8 Byte für den Header und (zumeist) weitere 16 Byte für die MAC ein. Für DTLS fallen weitere 8 Byte für die *Epoch* und *Seq-Nr* Felder an.

5.3.2.5 IP-Endpunkt oder Ende-zu-Ende Security

Eine Frage die sich für den Einsatz von (D)TLS in *adaptive AUTOSAR* stellt, ist auf welcher Ebene der Schutz angesetzt werden soll. (D)TLS ist für IP-Endpunkte (die Kombination von IP-Adresse und Port) definiert, während eine volle Ende-zu-Ende (E2E) Absicherung per Definition individuell je Service ansetzt. Trivial lässt sich (D)TLS für E2E-Schutz eines Service einsetzen, wenn jeder Service einen individuellen Port zugewiesen bekommt. Aufgrund der erheblichen Anzahl möglicher Services innerhalb des System würde dies jedoch zu einer sehr hohen Anzahl von TLS-Verbindung führen, wobei jede einzelne über einen eigenen TLS-Handschlag aufgebaut werden muss. Für eine korrekte E2E-Sicherheit müsste zudem jeder Service ein eigenes Zertifikat besitzen. Dies ist anhand des begrenzten sicheren Speichers und der harten Anforderung an die Startzeit des betrachteten Systems, in Kombination mit der eingeschränkten Rechenleistung, unrealistisch.

Folglich kommt aus Effizienzgründen lediglich eine Absicherung je IP-Endpunkt für das betrachtete System in Frage. Um dennoch eine möglichst hohes Sicherheitslevel für jeden einzelnen Service zu erreichen, sind die in Abschnitt 5.1 beschriebenen Anforderungen an eine sichere Plattform von entscheidender Bedeutung.

5.3.2.6 Ungeschützte Headerinformationen

(D)TLS zieht Ethernet und IP-Headerinformation nicht in die Berechnung der MAC mit ein, um die Schichtentrennung nicht zu durchbrechen. Dies erscheint zunächst als gefährliche Sicherheitslücke, ähnlich der in Abschnitt 5.2 diskutierten Schwäche des SecOC-Ansatzes. Denkbar wäre etwa die Manipulation der IP-Adresse einer SOME/IP Nachricht, z. B. um eine andere Service-Instanz anzusprechen oder Daten von zwei Instanzen zu vertauschen.
Bei einer korrekten Anwendung von (D)TLS ist ein solcher Angriff jedoch nicht erfolgreich, da zwei Service-Instanzen niemals dieselbe (D)TLS-Verbindung teilen.

- Sind die betreffenden Instanzen auf unterschiedlichen ECUs alloziert, so muss für das Abrufen von Daten bzw. Methoden eine individuelle Verbindung je ECU aufgebaut werden. Vertauscht ein Angreifer nun die IP- oder MAC-Adresse einer Nachricht, sodass die jeweils andere Instanz die Nachricht erhält, so wird diese die Nachricht verwerfen, da sie mit einem anderen Sitzungsschlüssel verschlüsselt ist.
- Befinden die Instanzen sich auf derselben ECU, müssen diese per SOME/IP Standard an unterschiedliche Ports gebunden sein. Da aber (D)TLS-Verbindungen individuell pro Port aufgebaut werden, scheitert der Angriff auch in diesem Fall, da wiederum ein unterschiedlicher Sitzungsschlüssel verwendet wird.

5.3.3 Analyse und sekundäre Sicherheitsattribute

Im folgenden Abschnitt werden die Sicherheitsgarantien von (D)TLS und die sekundären Sicherheitsattribute beschrieben, welche durch dessen Einsatz hinzukommen.

5.3.3.1 Analyse

(D)TLS erlaubt den Aufbau eines sicheren Kommunikationskanal zwischen zwei IP-Endpunkten, wobei sich beide Parteien während des Handschlags mithilfe von Zertifikaten authentifizieren. Außer den im Klartext übertragenen Informationen des Handschlags und der Headerinformationen werden sämtliche Nachrichten verschlüsselt übertragen. Der TLS-Header wird zusätzlich in die Berechnung der MAC miteinbezogen, wodurch der gesamte TLS-Frame zudem vom Empfänger authentifiziert werden kann. Dazu kommen temporäre Sitzungsschlüssel zum Einsatz, welche (bei Verwendung von DHE) die

perfect forward secrecy Eigenschaft erfüllen. Replay-Attacken verhindert das (D)TLS *Record Layer* durch das Einbeziehen der (expliziten oder impliziten) Sequenznummer in die MAC-Berechnung. Pakete mit einer bereits empfangenen Sequenznummer werden vom Empfänger automatisch verworfen und eine eventuelle Manipulation der Nummer wird spätestens bei Prüfung der berechneten MAC festgestellt und verworfen.

Der Einsatz von (D)TLS für die SOME/IP Unicast-Kommunikation kann sämtliche betrachtete Unicast-spezifische Angriffe (4.1-4.6, 5.2 und 5.3) verhindern, da (D)TLS die Sicherheitseigenschaften Integrität, Authentizität und Geheimhaltung garantiert. Die Sicherheit des Protokolls wird zudem - aufgrund dessen weiter Verbreitung - fortwährend analysiert [77] und verbessert.
Voraussetzung für diese Garantie ist, dass die privaten Schlüssel beider Kommunikationspartner geheim sind und ein CSPRNG zur Erzeugung der (einmalig) verwendeten Zufallszahlen eingesetzt wird.
Um den Verbindungsaufbau zu beschleunigen, existieren die *0-RTT* und *1-RTT* Versionen des Handschlags, um die Verbindung mithilfe eines vorherigen Sitzungsschlüssels wieder aufzunehmen. Dies ermöglicht einen erheblich beschleunigten Systemstart, da alle asymmetrischen Operationen zur Prüfung der Zertifikate bei *0-RTT* und *1-RTT* entfallen. Bei der Verwendung von *0-RTT* muss jedoch die verringerten Sicherheitsgarantien beachtet und geeignete Maßnahmen ergriffen werden, damit Replay-Attacken verhindert und hoch sensitive Daten nicht als *Early Data* übertragen werden.
Um die Sicherheitseigenschaft nicht gefährlich zu verringern, sollte ein einmal ausgehandelter Schlüssel nicht unbegrenzt oft wiederverwendet werden und der volle TLS-Handschlag regelmäßig wiederholt werden. Damit die Startzeit des Systems nicht in unregelmäßigen Abständen länger dauert als gewohnt, sollte ein solcher auffrischender Handschlag im laufenden Betrieb vorgenommen werden und nicht während des zeitsensitiven Systemstarts. Die Parteien wechseln dann zu einem vereinbarten Zeitpunkt auf die neue Verbindung, wobei dies für höhere Ebenen völlig transparent erfolgen kann. Analog zu dem Epochenwechsel von DTLS ist auch hierbei ein kurzer *Sliding-Window* Zeitbereich denkbar, in welchem Empfänger Nachrichten aus beiden Epochen bzw. Verbindungen akzeptieren, um unnötige Paketverluste zu vermeiden.

5.3.3.2 Sekundäre Sicherheitsattribute

Aus der vorherigen Analyse und der Bedingungen des Protokolls lassen sich direkt die folgenden sekundären Sicherheitsattribute bestimmen:

Zertifikate Alle potentiellen Teilnehmer der sicheren SOME/IP Kommunikation (alle physisch getrennten ECUs) müssen validierbare Zertifikate besitzen, damit eine wechselseitige Authentifizierung erfolgen kann. Dies verhindert Angriffe auf das System durch Einbringen einer (möglicherweise simulierten) ECU durch den Angreifer als neuen Kommunikationsteilnehmer. Ein solcher Angriff kann durch die legitimen ECUs erkannt werden, da der Angreifer kein gültiges Zertifikat vorweisen

kann.
Zur Prüfung der Zertifikate bietet sich der Einsatz einer herstellerspezifischen PKI (vgl. Abschnitt 2.3.2.7) an, wobei der Hersteller ein *Root*-Zertifikat besitzt, dessen öffentlicher Schlüssel ab Werk in allen Steuergeräten gespeichert ist. Zusätzlich erhält jedes (legitime) Steuergerät ein individuelles Zertifikat, welches durch das Herstellerzertifikat signiert und an ein Fahrzeug gebunden ist, z. B. durch Einbeziehen der eindeutigen Fahrzeug-Identifizierungsnummer in das entsprechende Zertifikat. Diese Bindung verhindert die Nutzung eines möglicherweise manipulierten Steuergeräts aus einem anderen Fahrzeug des selben Herstellers oder derselben Baureihe.
Da ein solcher Ansatz den Austausch einer defekten ECU unmöglich machen würde, wird hierfür ein zusätzlicher autorisierter Prozess benötigt. Ein möglicher Ansatz kann das Ausstellen eines neuen fahrzeuggebundenen Zertifikats für eine ECU durch den Herstellers sein, wenn der Einbau durch autorisierte Personen bzw. Werkstätten erfolgt. Die Übertragung des Zertifikats muss dabei höchsten Sicherheitsanforderungen genügen.
Zur Prüfung der Zertifikate während des TLS-Handschlags kann das empfangene Zertifikat anhand des gespeicherten öffentlichen Schlüssel des Herstellers geprüft werden. Für Diagnose in einer Werkstatt (oder für den Einbau neuer ECUs) kann der Hersteller ein kurzzeitig, gültiges Zertifikat ausstellen, welches Zugriff auf das System ermöglicht.
Durch den Einsatz von Attributzertifikaten können dabei die Rechte eines solchen Zertifikatsinhabers vom Hersteller genau festgelegt werden, z. B. könnte ein Diagnosezertifikat lediglich Zugang zu typischen Diagnosediensten gewähren, nicht jedoch auf Updatemechanismen oder datenschutzrelevante Daten des Fahrzeughalters.

Sicherer Speicher Die privaten Schlüssel von Server und Client müssen sicher gespeichert sein (vgl. Abschnitt 2.3.2.2). Zusätzliche Zertifikate, welche zur Validierung von Zertifikaten benötigt werden (z. B. ein Herstellerzertifikat), müssen ebenfalls schreibgeschützt gespeichert werden.
Zur Wahrung der *Perfect Forward Secrecy* Eigenschaft muss nach Abschluss einer Sitzung zudem der temporäre Sitzungsschlüssel und alle zu dessen Erzeugung genutzten Daten vollständig gelöscht werden.

CSPRNG Alle Systeme, welche (D)TLS in der vorgestellten Form einsetzen, müssen Zugang zu einem CSPRNG besitzen und diesen zur Erzeugung der verwendeten Zufallszahlen einsetzen (vgl. Abschnitt 2.3.2.4).

Sicheres Updateverfahren Trotz intensiver Analyse von (D)TLS sind dennoch neue Angriffe auf das Protokoll (und entsprechende Gegenmaßnahmen) zu erwarten. Zusätzlich könnten Fehler in der Implementierung entdeckt und korrigiert werden müssen. Eine weitere Gefahr stellen öffentlich gewordene oder abgelaufene Zertifikate dar, welche durch den Hersteller ersetzt werden müssen (*Revokation*).

Für einen solchen Fall ist ein schnelles, sicheres Updateverfahren unerlässlich - unabhängig von Service-Intervallen.

Einmal-Tickets Soll ein *0-RTT* Handschlag ausgeführt werden, muss der Server sichere Tickets an Clients ausstellen und sicherstellen, dass jedes Ticket nur einmal zur Fortsetzung der Verbindung eingesetzt werden kann. Außerdem sollte *0-RTT* nicht verwendet werden, um hochsensitive Informationen zu übertragen, da keine *perfect forward secrecy* für *0-RTT*-Daten besteht.

Wird eines oder mehrere dieser sekundären Sicherheitsziele verletzt, so können die Sicherheitseigenschaften des Verfahrens (zumindest teilweise) gebrochen werden.

5.4 Multicast-Kommunikation

SOME/IP setzt IP-Multicast in zwei prinzipiell getrennten Multicast-Semantiken ein: Event-Benachrichtigungen und Service Discovery (siehe Abbildung 5.3):

Eventgruppen Der zugrundeliegende Publish-Subscribe-Mechanismus impliziert eine *1:n* Kommunikation, mit einem festgelegten Sender und einer beliebigen Anzahl an Empfängern (siehe Teilbild 5.3a).

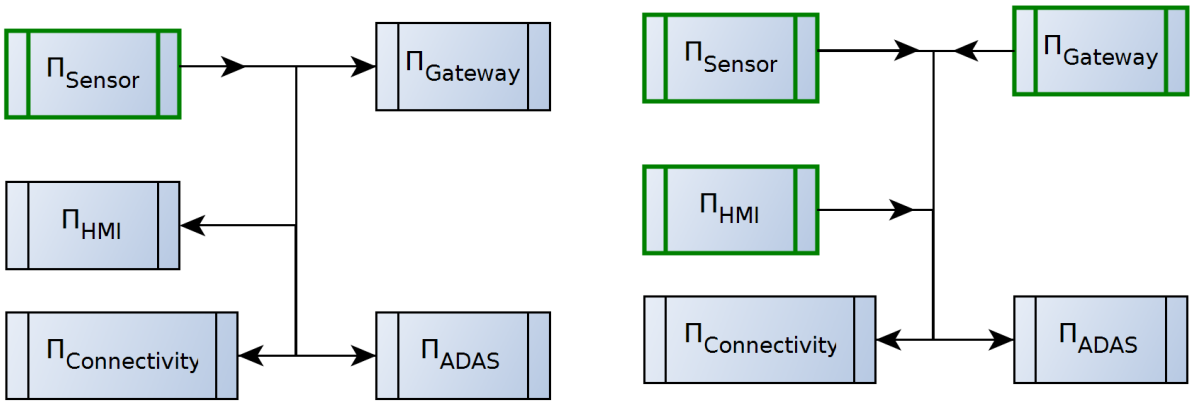
Service Discovery SOME/IP-SD ist ein allgemeines Multicast-Szenario mit *m:n* Kommunikation: eine beliebige Anzahl von Sendern kommuniziert mit einer beliebigen Anzahl von Empfängern (siehe Teilbild 5.3b).

IP-Multicast ist im Vergleich zur Singelcast-Anwendung weniger verbreitet und es ist komplexer eine effiziente Absicherung zu finden [78]. Im Folgenden werden die Schwierigkeiten unterschiedlicher Sicherheitseigenschaften von Multicast-Gruppen betrachtet.

5.4.1 Geheimhaltung und Schlüsselaustausch

Die Geheimhaltung der Gruppenkommunikation kann mithilfe eines von allen Teilnehmern geteilten symmetrischen Gruppenschlüssels effizient erreicht werden. Der Austausch dieses Schlüssels zwischen allen (berechtigten) Teilnehmern ist ein klassisches Problem der Kryptographie. Die drei klassischen Ansätze zur Lösung des Problems werden im Folgenden kurz besprochen.

Sicherer Kanal Besteht zwischen den Parteien bereits ein sicherer Kanal, so kann der Schlüssel über diesen ausgetauscht werden. Durch den Aufbau einer (D)TLS-Verbindung zwischen zwei Parteien kann ein symmetrischer Schlüssel zur Verschlüsselung einer Multicast-Kommunikation folglich sicher ausgetauscht werden,



(a) SOME/IP-Events verwenden eine *1:n* Multicast-Semantik und besitzen einen festgelegten Sender und eine beliebige Anzahl an Empfängern.

(b) Die Service Discovery verwendet dagegen eine *m:n*-Kommunikation, bei der eine beliebige Anzahl an Sendern mit einer beliebigen Anzahl an Empfängern kommunizieren.

Abbildung 5.3: SOME/IP Multicast-Semantiken: *1:n*-Events und *m:n*-Service Discovery. Die jeweils aktiven Sender sind grün markiert.

da (D)TLS die Sicherheitseigenschaften Authentizität und Geheimhaltung garantiert. Dieser Ansatz ist für eine *1:n* Kommunikation mit Anmeldung geeignet, da einem Teilnehmer der Gruppe nach dessen Beitritt der Schlüssel über die ohnehin benötigte abgesicherte Verbindung übertragen werden kann. Für ein *m:n* Kontext skaliert der Ansatz jedoch schlecht.

Trusted-Third-Party Die Teilnehmer besitzen keinen sicheren Kanal zueinander, jedoch kann jeder Teilnehmer eine sichere Verbindung zu einer vertrauenswürdigen Drittpartei aufbauen. In einer solchen Konstellation kann die dritte Partei einen Schlüssel wählen und über die jeweiligen gesicherten Kanäle an die einzelnen Teilnehmer verteilen. Das Verfahren skaliert gut für eine *m:n* Situation, da lediglich ein Nachrichtenpaar zwischen jedem Teilnehmer und der Drittpartei ausgetauscht werden muss. Problematisch ist die Voraussetzung der vertrauenswürdigen Drittpartei, welche einen Single-Point-of-Failure für die Sicherheit des Systems darstellt und die Notwendigkeit von n sicheren Kommunikationskanälen.

Multi-Party DH Die Verallgemeinerung des Diffie-Hellman Schlüsselaustauschs kann zur Erzeugung eines Multicast-Sitzungsschlüssels über ungesicherte Kanäle verwendet werden [79]. Dies erfordert insgesamt $O(n)$ übertragene Multicast-Nachrichten und $O(n^2)$ Faktorisierungen. Der Ansatz skaliert gut für das allgemeine *m:n* Multicast-Szenario, ist jedoch rechenintensiver als die vorherigen Ansätze.

5.4.2 Dynamische Gruppenmitgliedschaft

Besondere Vorkehrungen müssen für dynamische Multicast-Gruppen getroffen werden. Sowohl der Beitritt als auch das Verlassen eines Gruppenmitglieds kann die aktuellen bzw. zukünftigen Sicherheitseigenschaften der Gruppenkommunikation beeinflussen. Für das betrachtete System ist offensichtlich, dass sowohl Eventgruppen als auch das SOME/IP-SD Protokoll dynamische Gruppen verwenden. In beiden Fällen ist ein effizienter Gruppenbeitritt entscheidend, um den Systemstart kurz zu halten.

5.4.2.1 Gruppenbeitritt

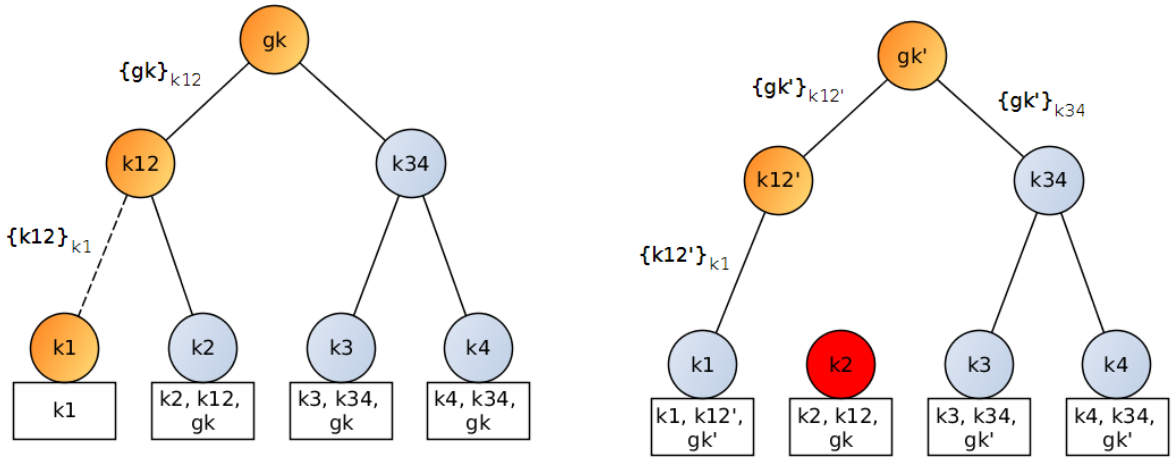
Der Beitritt eines neuen Gruppenmitglieds kann die Geheimhaltung zuvor gesendeter Nachrichten (gegenüber dem neuen Mitglied) gefährden. Hat das neue Mitglied zuvor gesendete Nachrichten aufgezeichnet und wurde der symmetrische Gruppenschlüssel seither nicht geändert, so kann dieser offensichtlich alle zuvor aufgezeichneten Nachrichten nach dem Erhalt des Schlüssels entschlüsseln.
Ist dies nicht erwünscht, so muss vor dem Beitritt des neuen Mitglieds ein neuer Gruppenschlüssel gewählt und an alle Mitglieder (inklusive dem hinzugekommenen) verteilt werden. Wird der jeweils aktuelle Gruppenschlüssel von einem zentralen Knoten verwaltet, so ist dies trivial möglich (vgl. Abschnitt 5.4.1). Bei einem verteilten Verfahren (z. B. Multi-Party Diffie-Hellman) muss dagegen typischerweise ein komplett neuer Schlüsselaustausch gestartet werden.

5.4.2.2 Verlassen der Gruppe

Damit die Sicherheitseigenschaften der Multicast-Gruppe nach dem Verlassen eines Mitglied gewahrt bleiben, muss unter allen verbleibenden Mitgliedern ein neuer Gruppenschlüssel verteilt werden, ohne dass das ausgeschiedene Mitglied diesen neuen Schlüssel erhält. Insbesondere bedeutet dies, dass ein neuer Schlüssel nicht mit dem vorherigen verschlüsselt übertragen werden darf.
Naiv lässt sich dieses Problem nur durch das Erstellen einer neuen Multicast-Gruppe lösen - inklusive dem jeweiligen Schlüsselaustauschverfahren zur Erstellung eines neuen Gruppenschlüssels.

5.4.2.3 Schlüsselbäume

Eine Alternative für die Verwaltung eines symmetrischen Gruppenschlüssels im Fall dynamischer Gruppenmitgliedschaft bieten sogenannte Schlüsselbäume an [78][80]. Dazu wird von einem Gruppenmeister ein binärer Baum von Schlüsseln erstellt, wobei die Wurzel des Baums identisch zu dem aktuellen Gruppenschlüssel ist. Jeder Knoten



(a) *Join:* Knoten 1 tritt der Gruppe bei und soll den Gruppenschlüssel gk und alle KEKs entlang des Pfads von Knoten 1 zur Wurzel (orange markiert) erhalten ($k12$ und gk).

(b) *Leave:* Knoten 2 verlässt die Gruppe, woraufhin betroffenen KEKs geändert werden müssen (orange markiert).

Abbildung 5.4: Ein Schlüsselbaum zur Verwaltung eines symmetrischen Gruppenschlüssels. Knoten bezeichnen logische KEKs, Blätter die individuellen KEKs der Gruppenmitglieder. Rechtecke unter Blättern enthalten alle Schlüssel die der entsprechende Knoten kennt.

des Baums stellt einen KEK (engl. *Key Encryption Key*) dar, welcher zur Verschlüsselung des jeweiligen Elternknotens eingesetzt wird. Die Blätter des Baums sind identisch zu den Empfängern, wobei jeder Teilnehmer einen individuellen KEK besitzt. Diese individuellen Schlüssel werden einem Mitglied beim Gruppenbeitritt über einen sicheren Unicast-Kanal übermittelt. Jeder Empfänger hält zudem alle KEKs entlang des Pfads vom eigenen Blattknoten zur Wurzel - insgesamt folglich maximal $\log_2(n) + 1$ Schlüssel.

Der Gruppenmeister sendet geänderte Schlüssel gesammelt in einer *reKey* Multicast-Nachricht. Ein geänderter Schlüssel wird jeweils mit den Schlüsseln seiner Kinderknoten verschlüsselt übertragen. Wird im Beispiel (siehe Abbildung 5.4) der Gruppenschlüssel gk zu gk' geändert, so wird die *reKey*-Nachricht die Einträge $\{gk'\}^s_{k12}$ und $\{gk'\}^s_{k34}$ enthalten. Diese *reKey*-Nachricht hat eine maximale Länge von $2\log_2(n)$, wobei diese noch um den Faktor 2 verringert werden kann, indem die KEKs nicht vom Gruppencontroller beliebig gewählt, sondern durch eine Einwegfunktionen generiert werden [80].

Für den Beitritt bzw. das Verlassen einer Gruppe unter Verwendung eines Schlüsselbaums, werden die folgenden Operationen durchgeführt (vgl. Abbildung 5.4):

Join Ein neues Blatt wird in den Baum eingefügt (vgl. Abbildung 5.4a). Der neue beigetretene Knoten 1 erhält seinen individuellen Schlüssel k_1 über eine gesicherte Unicast-Verbindung. Anschließend kann der Gruppenmeister den Gruppenschlüssel und alle KEKs entlang des Pfads zwischen Wurzel und Knoten 1 entweder

direkt als Teil der gesicherten Unicast-Nachricht oder als Multicast-Nachricht übertragen. Im letzteren Fall erfolgt dies entsprechend der zuvor beschriebenen Verschlüsselungsvorschrift. Im Beispiel (5.4a) würde der Gruppenmeister dazu die folgende Nachricht übertragen: $\langle \{k12\}_{k1}, \{gk\}_{k12} \rangle$. Knoten 1 kann $k12$ mit seinem individuellen Schlüssel entschlüsseln, woraufhin $k12$ zur Entschlüsselung des Gruppenschlüssels gk eingesetzt werden kann.
Soll der neu hinzugekommene Knoten nicht in der Lage sein, vorherige Gruppenkommunikation entschlüsseln zu können (*backwards secrecy*), so kann vor dessen Beitritt der Gruppenschlüssel und betroffene KEKs (im Beispiel $k12$) geändert werden.

Leave Analog zu einem *Join* mit *backwards secrecy* werden beim Verlassen eines Knotens alle KEKs entlang des Pfads zwischen dem betroffenen Knoten (im Beispiel ist dies Knoten 2) und der Wurzel erneuert. Einzige Ausnahme ist, dass der KEK des direkten Elternknotens (im Beispiel Knoten 12) nur mit dem verbleibenden Kinderknoten verschlüsselt. Dies bedeutet, dass die *reKey*-Nachricht insbesondere den Eintrag $\{k12'\}_{k2}$ *nicht* enthält. Ohne diesen ist der verlassende Knoten nicht in der Lage den neuen KEK des Elternknotens zu entschlüsseln, und folglich kann dieser auch den neuen Gruppenschlüssel nicht entschlüsseln. Im Beispiel (5.4b) würde die *reKey*-Nachricht wie folgt aussehen: $\langle \{gk'\}^s_{k12'}, \{gk'\}^s_{k34}, \{k12'\}^s_{k1} \rangle$.

Offensichtlicher Nachteil des Verfahrens ist der Einsatz eines zentralen Knotens, zur Verwaltung des Gruppenschlüssels, der damit einen *single point of failure* für des Systems darstellt. Dieser Nachteil ist jedoch lediglich für die SOME/IP-SD Phase von Bedeutung, da die *1:n* Eventgruppen-Kommunikation bei einem Ausfall der anbietenden ECU auch der zugehörige Schlüsselbaum nicht länger vonnöten ist.

5.4.3 Integrität und Authentifizierung

Die Integritäts- und Authentifizierungskriterien für Multicast-Kommunikation können jeweils in zwei unterschiedlich starken Ausprägungen erreicht werden:

Starke Integrität Diese Sicherheitseigenschaft garantiert, dass weder ein externer Angreifer (ohne Kenntnis des Gruppenschlüssels), noch ein Mitglied der Gruppe eine gesendete Nachricht manipulieren kann. Diese Eigenschaft kann nicht durch den klassischen Einsatz eines symmetrischen Gruppenschlüssel zur Berechnung der MAC für gesendete Nachrichten erreicht werden, da alle Gruppenmitglieder im Besitz des Schlüssels sind. Dies erlaubt Gruppenmitgliedern eine Nachricht zu manipulieren und eine dazu passende MAC mit dem Schlüssel zu berechnen. Ein einfacher Check der MAC kann eine solche Manipulation nicht erkennen.

Schwache Integrität Diese abgeschwächte Form der Integritätseigenschaft garantiert lediglich die Integrität der Nachricht gegenüber einem externen Angreifer, nicht jedoch für Gruppenmitglieder. Dies kann durch die klassische Verwendung eines

symmetrischen MAC erreicht werden. Ein externer Angreifer, der nicht im Besitz des Gruppenschlüssels ist, kann für eine manipulierte Nachricht keine passende MAC berechnen. Eventuelle Änderungen können von den Empfängern somit beim Test der MAC erkannt werden.

Senderauthentifizierung Diese Sicherheitseigenschaft (engl. *Source Authentication*) ist die logische Erweiterung des Authentifizierungskriteriums auf Multicast-Kommunikation. Senderauthentifizierung garantiert, dass eine empfangene Nachricht tatsächlich von dem vorgeblichen Absender gesendet und nicht manipuliert wurde.
Naiv lässt sich diese Garantie lediglich durch Signieren jeder einzelnen gesendeten Nachricht mit dem privaten Schlüssel des Absenders erreichen. Dies erfordert jedoch erhebliche Rechenleistung, sowohl auf Seiten des Senders als auch von allen Empfängern.
Zudem stellt sich bezüglich SOME/IP die Frage, ob verschiedene Services (und Service-Instanzen) innerhalb einer ECU als prinzipiell getrennte Sender interpretiert werden sollten (d. h. jeder Service und jede Service-Instanz besitzt eine eigene Senderauthentifizierung) oder ob eine ECU-gebundene Senderauthentifizierung ausreichend ist. Analog zu (D)TLS bietet sich eine Senderauthentifizierung je (IP, Port)-Tupel an. Damit kann ein geringer Ressourcenbedarf und gleichzeitig eine eindeutige Authentifizierung erreicht werden, insbesondere auch für alle Instanzen eines Service, unabhängig ob dieser auf derselben oder verschiedenen ECUs allokiert sind (vgl. Argumentation zur Sicherheit von (D)TLS im Abschnitt 5.3.2.6).

Gruppenauthentifizierung Hierbei wird lediglich garantiert, dass eine empfangene Nachricht von einem Mitglied der Gruppe gesendet wurde. Da diese Garantie nicht ausschließt, dass ein Mitglied der Gruppe sich für ein anderes Gruppenmitglied ausgibt, stellt Gruppenauthentifizierung eine (erheblich) schwächere Sicherheitseigenschaft als Senderauthentifizierung dar.
Die Anwendung eines symmetrischen Gruppenschlüssels (vgl. 5.4.1) zur Berechnung einer MAC der gesendeten Nachricht garantiert offensichtlich Gruppenauthentifizierung, nicht jedoch Senderauthentifizierung.

Gruppenauthentifizierung bzw. die schwache Integritätseigenschaft sind nur dann ausreichend, wenn allen Gruppenmitgliedern vertraut werden kann. Dies ist für die Szenarien Applikations- und Netzwerkkontrolle der Fall, vorausgesetzt die Trusted-Plattform Annahme ist erfüllt, nicht jedoch für einen Angreifer mit voller ECU-Kontrolle. Senderauthentifizierung bzw. starke Nachrichtenintegrität kann die gesendeten Nachrichten ehrlicher Teilnehmer dagegen auch dann schützen, wenn ein Mitglied der Gruppe vollständig kompromittiert ist.
Senderauthentifizierung effizient zu implementieren ist jedoch nicht trivial, da der klassische Einsatz symmetrischer MACs ungenügend, die Verwendung digitaler Signaturen jedoch zu rechenintensiv ist [78][81][82].

5.4.4 Zwischenstand

Durch eine Kombination der bisher diskutierten Verfahren lassen sich die folgenden Sicherheitseigenschaften für das betrachtete System erreichen:

1. Unicast: Durch die Anwendung von (D)TLS können die Sicherheitseigenschaften Integrität, Authentizität, Geheimhaltung und Freshness erfüllt werden.
   Der Aufbau des sicheren (D)TLS-Kommunikationskanals muss nur einmal (z. B. bei Systemstart) in der vollen Länge erfolgen. Nach einem vorübergehenden Abbau der Verbindung (z. B. nach einer Standby-Phase eines Teilnehmers) kann ein verkürzter Handschlag durchgeführt werden, um die sichere Kommunikation fortzusetzen.

2. *1:n*-Multicast: Das Publish-Subscribe-Konzept für Eventgruppen kann ohne zusätzlichen Kommunikationsoverhead abgesichert werden, sodass die Geheimhaltungseigenschaft erfüllt werden kann. Dazu überträgt der Sender den verwendeten Schlüssel an die abonnierenden Clients nach deren Beitritt zur Gruppe. Dieser Austausch muss mithilfe einer abgesicherten Verbindung erfolgen (z. B. mittels (D)TLS). Eine genauere Betrachtung des Beitritts erfolgt in Kapitel 6.3.2.2.

3. Dynamische Multicast-Gruppen: Mithilfe eines aufgespannten Schlüsselbaums kann eine dynamische Gruppenmitgliedschaft verwaltet werden. Insbesondere das Verlassen der Gruppe kann so effizient implementiert werden.
   Für die Verteilung eines Gruppenschlüssels kann ein zentraler Knoten oder ein verteiltes Verfahren zur Aushandlung eines gemeinsamen Schlüssels eingesetzt werden. Dieser kann, wie bereits besprochen, lediglich die Geheimhaltung und Gruppenauthentizität gesendeter Nachrichten sicherstellen, nicht jedoch Senderauthentizität.

Ein Vergleich mit den in Kapitel 4.2.5 beschriebenen Sicherheitszielen zeigt, dass die folgenden Eigenschaften noch nicht erreicht sind und eine weitere Untersuchung erfordern:

1. *1:n* Multicast: Für ein höheres Sicherheitsniveau der Multicast-Kommunikation sollte ein Konzept zur Erfüllung von Senderauthentizität und des starken Integritätskriteriums gefunden werden. Dies erlaubt die Wahrung der Sicherheitseigenschaften auch in der Gegenwart eines Angreifers mit voller ECU-Kontrolle über eines der Empfänger.

2. *n:m* Multicast: Das Konzept zur Absicherung der *1:n* Kommunikation lässt sich nicht auf den allgemeinen Multicast Fall übertragen. Da das SOME/IP-SD Protokoll *m:n* Multicast einsetzt, wird ein Ansatz zu dessen Absicherung benötigt. Ein solcher Ansatz sollte insbesondere Senderauthentizität und das starke Integritätskriterium erfüllen können, da beide Eigenschaften essentiell für die Korrektheit des gesamten Systems sind.

5.5 Verfahren für sicheren Multicast

In Abschnitt 5.4.4 wurden eine Reihe offener Sicherheitseigenschaften beschrieben, alle in Bezug auf die Absicherung von Multicast-Kommunikation. Im folgenden Kapitel werden verschiedene Multicast-Sicherheitsverfahren betrachtet, welche insbesondere die Eigenschaft der Senderauthentifizierung garantieren. Diese ist entscheidend, um eine sichere Gruppenkommunikation auch im Fall eines vollständig kompromittierten Gruppenmitglieds zu gewährleisten.

5.5.1 Hybride Verfahren

Zentrales Problem für das Erreichen von Senderauthentifizierung ist, dass die Berechnung digitaler Signaturen für jede gesendete Nachricht zu rechenintensiv ist. Ein Reihe von Ansätzen existieren, welche die rechenintensive Bestimmung einer digitalen Signatur über mehrere Nachrichten eines Blocks hinweg verteilen [82][83][84]. Dazu wird für jede Nachricht ein Hashwert berechnet und eine Signatur über die Konkatenation dieser Hashwerte gebildet, anstatt über die vollen Nachrichten. Ein Empfänger kann die signierten Hashwerte nach dem Empfang mit den neu berechneten Werten vergleichen und so eventuelle Manipulationen entdecken.
Inhärentes Problem dieser Konstruktion ist, dass der Verlust eines Pakets die Authentifizierung des gesamten Nachrichtenblocks unmöglich macht. Um dem entgegen zu wirken, setzen hybride Verfahren zumeist Fehler korrigierende Codes ein, welche in der Lage sind, eine gewisse Zahl verlorener Pakete zu kompensieren [82][84].
Zudem erfordert ein solches Verfahren, dass entweder der Sender oder Empfänger alle Nachrichten eines Blocks puffert, da die Authentizität der Nachrichten erst verifiziert werden kann, wenn alle Pakete eines Blocks empfangen wurden. Dies führt - in Abhängigkeit von der Blockgröße - zu einer erheblichen Erhöhung der Latenz.

5.5.1.1 Hybrides Verfahren nach Pennetrat und Molva

Pennetrat und Molva beschreiben ein effizientes und robustes Verfahren zur Authentifizierung eines Datenstroms unter Einsatz von ECC, dessen Eigenschaften sich durch eine geeignete Parameterwahl an ein gegebenes System anpassen lässt [82]. Dabei wird ECC nicht, wie typischerweise, zur Wiederherstellung der Datenpakete selbst eingesetzt, sondern zur Wiederherstellung derjenigen Fragmente der Authentifizierungsinformation, welche Teil der verlorenen Pakete sind. Dies ermöglicht einem Empfänger alle Pakete eines Blocks zu authentifizieren, selbst wenn nicht alle empfangen wurden. Gleichzeitig wird die Größe der einzelnen ECC-Fragmente deutlich reduziert.
Dazu werden b Pakete $P_1, ...P_b$ eines Datenstroms zu einem Block B zusammengefasst, wobei die Länge eines Pakets aus Sicht des Verfahrens beliebig ist. Für jedes Paket P_i des Blocks wird ein Hashwert $h_i = H(P_i)$ berechnet. Diese Hashwerte werden zu einem

Bitstring X konkateniert, welcher als Eingabe für einen separierbaren *Erasure*-Code (siehe Abschnitt 2.3.3) dient, wobei $\bar{X}$ die Paritätssymbole des Codes bezeichnen. Anschließend wird über X eine digitale Signatur $\sigma = \text{Sign}(H(X))$ gebildet. Anschließend wird über der Konkatenation $(\bar{X}, \sigma)$ erneut ein *Erasure*-Code gebildet, dessen Paritätssymbole mit $\bar{Y}$ bezeichnet werden. Das resultierende Codewort $(\bar{X}, \sigma, \bar{Y})$ wird in b Fragmente τ_i gleicher Länge aufgeteilt und jedem ursprünglichen Paket P_i des Blocks wird ein solches Fragment angehängt.
Diese komplexe Konstruktion wird in Abbildung 5.5 zusammengefasst. Empfänger puf-

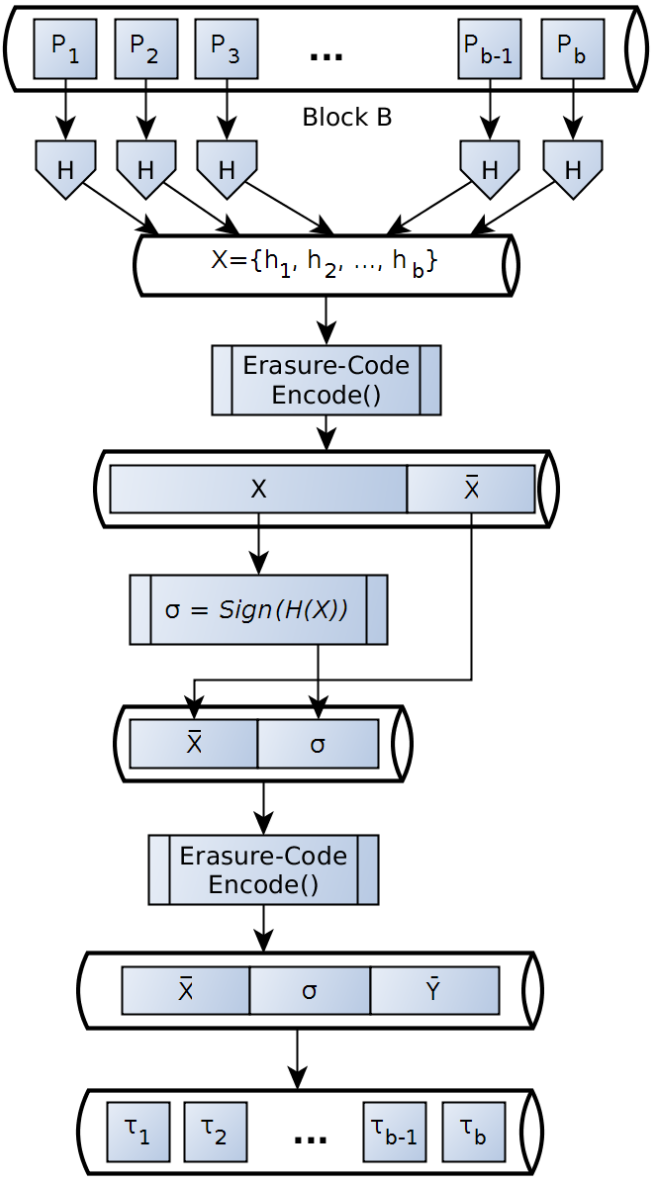


Abbildung 5.5: Konstruktion der Authentifizierungsfragmente τ des hybriden Verfahrens [82].

fern ihrerseits alle Nachrichten eines Blocks, bevor sie mit der Authentifikation beginnen. Dazu prüfen sie, ob ein Paketverlust aufgetreten ist. Ist dies der Fall, werden die Hashwerte der fehlenden Pakete mithilfe der entsprechenden ECC-*Decode* Operation wiederhergestellt, solange die Anzahl der verlorenen Pakete nicht zu hoch ist. Sollten keine Verluste aufgetreten sein, oder alle verlorenen Hashwerte wiederhergestellt worden sein, so kann der Empfänger die Signatur der Hashwerte prüfen, um so die Authentizität der Nachricht sicherzustellen.

Insgesamt werden die folgenden Operation für jeden vollen Block durchgeführt: zwei *ECC-Encode*, b Berechnungen einer kryptographisch sicheren Hashfunktion und eine digitale Signatur. Empfängerseitig wird die selbe Anzahl der entsprechenden Operationen (*Verify* statt *Sign* und *ECC-Decode* statt *ECC-Encode*) ausgeführt. Dabei können die ECC-Operationen entfallen, falls keine Paketverluste aufgetreten sind.
Die entscheidenden Parameter des Verfahrens sind die Blocklänge b und die angenommene Verlustrate pro Block p, welche toleriert werden kann. Die Verlustrate pro Block kann ungefähr (mithilfe einer auf Markov-Ketten basierenden Simulation) aus der angenommen Paketverlustrate und der Gesamtlänge des Blocks bestimmt werden. Aus diesen Parametern kann ein entsprechender *Erasure*-Code erzeugt werden, damit mit einer hohen Wahrscheinlichkeit alle empfangenen Nachrichten eines Blocks authentifiziert werden können.

5.5.1.2 Analyse und Anforderungen

Das Verfahren von Pennetrat und Molva erzeugt einen robusten, senderauthentifizierten, Datenstrom durch die Anwendung von digitalen Signaturen auf Blöcke von gepufferten Paketen. Durch Anpassung der Parameter an erwartete Systemparameter (insb. die erwartete Verlustrate von Paketen und die Anzahl der zu puffernden Pakete) kann die Robustheit und der Overhead (sowohl im Bezug auf Puffergröße als auch der benötigten Rechenleistung) des Verfahrens variiert werden. Da zur Authentifizierung digitale Zertifikate des Senders verwendet werden, erfüllt das Verfahren zudem die Sicherheitseigenschaft der Verbindlichkeit. Dies macht das Verfahren besonders interessant für rechtlich relevante Einsatzzwecke, z. B. für Blackbox-Anwendungen oder Mautsysteme. Ein weiterer attraktiver Aspekt des Verfahrens ist, dass Empfänger (mit Ausnahme des öffentlichen Schlüssels des Senders) keinerlei zusätzliche Informationen benötigen, um Nachrichten zu empfangen und authentifizieren. Ebenfalls ist das Verfahren ohne Variation für *m:n* Kommunikation einsetzbar. Zuletzt profitiert das Verfahren von verlustfreier Übertragung, da in diesem Fall für den Empfänger keinerlei ECC-Operationen anfallen.

Das Verfahren stellt die Annahme auf, dass der Index eines Pakets innerhalb eines Blocks bekannt ist (z. B. durch die Nutzung einer Sequenznummer). Dies ist notwendig, um eventuelle Paketverluste zu erkennen und für die korrekte Durchführung von *ECC-Operationen*. Zudem wird angenommen, dass allen Empfängern der öffentliche Schlüssel des Senders bekannt ist. Dies ist für das gegeben System (typischerweise) möglich, da

die Hardwarekomponenten nur selten gewechselt werden. Dennoch ist ein Verfahren notwendig um öffentliche Schlüssel gesichert auszutauschen und zu übertragen, für den Fall dass eine Komponente ausgetauscht wird oder ein externer Kommunikationspartner an der Kommunikation teilnehmen will.
Aufgrund der Verwendung asymmetrischer Kryptographie zur Authentifizierung, ECC-Operationen und der Pufferung eines ganzen Blocks stellt sich die Frage nach der Performanz des Verfahrens. Diese lässt sich nur in einem Modellversuch bestimmen, für ein festes Parameterpaar b, p und reelle Zeitwerte für die Berechnung verschiedener kryptographischer Operationen.
Aus der Konstruktion ist jedoch offensichtlich, dass das Verfahren ein Abwägen zwischen Pufferspeicher und Rechenaufwand ermöglicht. Da nur eine Signatur pro Block berechnet wird, ist ein größerer Block effizienter im Bezug auf Rechenzeit, da die rechenintensive asymmetrische Operation *Sign(x)* verhältnismäßig selten durchgeführt werden muss. Zugleich bedeutet ein großer Block jedoch auch den Bedarf von erheblichem Pufferspeicher und eine entsprechend hohe Verzögerungszeit zwischen dem Absenden einer Nachricht und deren Authentifizierung.

5.5.2 l-Bit MAC

Ein möglicher Ansatz zur Lösung des Problems mithilfe von symmetrischer MACs ist der Aufbau eines Schlüsselbaums [78]. Dazu erzeugt der Sender l-symmetrische Schlüssel und verteilt an jeden Empfänger eine zufällig, gleichverteilte Teilmenge R_l. Der Sender berechnet für jeden gehaltenen Schlüssel l einen MAC für die gesendete Nachricht und hängt diese an die Nachricht an. Jeder Empfänger validiert alle MACs für die er ein passenden Schlüssel hält und akzeptiert, falls alle betrachteten MACs valide sind. Das Argument für Senderauthentifizierung ist dessen Kenntnis aller Schlüssel l und der geringen Wahrscheinlichkeit, dass zwei Empfänger dieselbe Teilmenge an Schlüsseln besitzen. Die Sicherheit des Ansatzes basiert auf zwei Parametern: q und ω.
Dabei gibt q die Wahrscheinlichkeit an, dass ein Angreifer eine zufällige Nachricht authentifizieren kann - ohne jedoch a priori zu wissen ob eine geratene MAC valide ist oder nicht. Die Autoren bezeichnen diese abgeschwächte Authentizität-Sicherheitseigenschaft als *q-per-Message unforgability* [78] und argumentieren, dass für die meisten Systeme ein kleines, aber nicht vernachlässigbares q ausreichend ist.
Mit ω wird die Anzahl korrumpierter Empfänger beschrieben, für welche die Sicherheitseigenschaft des Verfahrens erfüllt ist. Korrumpierte Empfänger teilen insbesondere alle ihnen bekannte Schlüssel.
Aus diesen Werten kann die Anzahl der benötigten Schlüssel l berechnet werden [78]:

$$|l| = e(\omega + 1)\ln\left(\frac{1}{q}\right) \tag{5.1}$$

e Die Eulersche Zahl e.

ln(x) Der natürliche Logarithmus.

Interessante Eigenschaft des Ansatzes ist, dass die Anzahl (nicht korrumpierter) Empfänger beliebig groß sein kann, ohne Einfluss auf die Sicherheitseigenschaft oder den Overhead des Verfahrens. Zudem kann das Verfahren einfach auf ein *n:m* Szenario übertragen werden, wobei die Anzahl der verwendeten Primärschlüssel l und die Sicherheitsparameter u und ω identisch zum *1:n* Szenario bleiben. Dazu wird jedem primären Schlüssel l eine pseudo-zufälligen Funktion (engl. pseudo random function, PRF) $f_k(x)$ zugeordnet. Alle Sender u erhalten ein Satz $l'_i = f_{l_i}(u)$ von sekundären Schlüsseln, welche durch die Anwendung der PRF auf die originalen Schlüssel und die Sender-ID von u erzeugt werden. Ein Sender berechnet für jedes gesendete Paket l MACs mit seinen sekundären Schlüsseln l'. Ein Empfänger kann die Authentizität der Nachricht bestimmen, indem er mittels der ID des Senders und den ihm bekannten Schlüsseln eine Teilmenge der sekundären Schlüssel l' von u berechnet. Stimmen alle validierbaren MACs überein, so akzeptiert der Sender die Nachricht als authentisch.

Um den erheblichen Overhead des Verfahrens (l MACs je Nachricht) zu reduzieren, können *single-bit* MACs verwendet werden [78]. Eine Koalition kann eine solche MAC lediglich mit einer Chance von $\frac{1}{2}$ raten. Aufgrund der hohen Anzahl an verwendeten Schlüsseln kann dennoch eine (theoretisch beliebig) hohe Sicherheitsstufe erreicht werden. Ein weiterer Performanzgewinn kann durch die Verwendung eines MAC-Verfahren erreicht werden, welches effizient MACs auf fixen Daten aber unterschiedlichen Schlüssel berechnen kann.

5.5.2.1 Analyse und Anforderungen

Das Verfahren nutzt l symmetrische MACs um ein abgeschwächte Form der Senderauthentifizierung zu erreichen. Die Anzahl der benötigten Schlüssel ist dabei sehr groß, wenn eine hohe Sicherheitsstufe erreicht werden soll - selbst wenn die angenommene Koalitionsgröße ω klein ist. Alle Schlüssel müssen vom Sender gespeichert werden, und jeder Teilnehmer muss eine (nicht triviale) Teilmenge dieser Schlüssel speichern - und zuvor über einen gesicherten Kanal vom Sender erhalten.

Für sicherheitskritische Anwendungen ist das Verfahren aufgrund der nicht vernachlässigbar kleinen Wahrscheinlichkeit für einen Angreifer eine Nachricht fälschlicherweise zu authentifizieren nicht geeignet. Zudem werden für den effizienten Einsatz des Verfahrens nicht-standardisierte MAC-Verfahren und erheblicher sicherer Speicherplatz benötigt.

5.5.3 TESLA

Ähnlich dem zuvor vorgestellten Verfahren nutzt TESLA ausschließlich symmetrische MACs, um eine Multicast-Senderauthentifizierung zu erreichen [81][85][86][87].

Die High-Level Idee der TESLA-Methodik sieht die Verwendung eines symmetrischen Schlüssels zur Berechnung einer MAC für gesendete Nachrichten vor. Diese werden im Folgenden zur besseren Unterscheidbarkeit als *TMAC* bezeichnet. Der verwendete Schlüssel ist dabei (zunächst) nur dem Absender bekannt. Empfänger können folglich den *TMAC* einer empfangenen Nachrichten nicht sofort validieren und müssen die Nachricht puffern, wobei zusätzlich der jeweilige Empfangszeitpunkt gespeichert wird. Erst nach Ablauf einer definierten Zeitspanne veröffentlicht der Sender den zuvor verwendeten Schlüssel. Mithilfe der gespeicherten Empfangszeit und einer bekannten Obergrenze für die maximale Netzwerkverzögerung, können Empfänger den Sendezeitpunkt einer Nachricht bestimmen. Liegt dieser vor dem Zeitpunkt, zu welchem der verwendete Schlüssel veröffentlicht wird, so kann kein Angreifer die *TMAC* der Nachricht erzeugt haben. Folglich kann die Nachricht nur von dem tatsächlichen Sender übertragen worden sein.

5.5.3.1 One-Way Key Chain

TESLA setzt eine sogenannte *One-Way Key Chain* ein, um selbst-authentifizierende Schlüssel zu erzeugen. Damit wird eine Kette von Werten einer Einwegfunktion beschrieben, welche als Basis für die Erzeugung von kryptographischen Schlüsseln genutzt werden (siehe Abbildung 5.6). Jedes Element der Kette wird einem Intervallindex i zugeordnet. TESLA-Sender unterteilen die Zeit in Intervalle I_i der fixen Länge T_{int}. Anhand einer definierten Startzeit T_0 kann das aktuelle Intervall $T_i = T_0 + T_{int} \cdot i$ bestimmt werden und das entsprechende Kettenelement h_i verwendet werden. Prinzipiell könnte das Kettenelement direkt als Schlüssel k_i verwendet werden, jedoch sollte Schlüsselmaterial in der Regel nicht für mehrere verschiedene kryptographische Operationen eingesetzt werden, weshalb die Schlüssel mithilfe einer (bekannten) PRF aus den Hashwerten abgeleitet werden. Als PRF kann dazu beispielsweise eine KDF eingesetzt werden, analog zu einem erneuten Schlüsselwechsel bei (D)TLS. Der so erzeugte Schlüssel k_i wird zur Berechnung der TMACs aller Nachrichten eingesetzt, welche während dem Intervall I_i übertragen werden.

Ein Sender wählt zur Berechnung der Kette einen zufälligen Wert h_n und wendet auf diesen wiederholt die Einwegfunktion H an - dies kann z. B. eine (kryptographisch sichere) Hashfunktion sein. Die letzte Anwendung von H in der Kette ergibt den Wert h_0, welcher als (initialer) *Commit* der Kette bezeichnet wird. Der Commit muss allen potentiellen Empfängern bekannt sein (siehe Abschnitt 5.5.3.4) und dient zur Authentifizierung von Kettenelementen und den daraus berechneten Intervallschlüsseln. Ist ein Commit und der Index i eines beliebigen Kettenelements h_i bekannt (z. B. nach dessen Veröffentlichung), so kann die Authentizität des Elements anhand des Commit geprüft werden. Dazu berechnet der Empfänger $H^i(h_i)$, die i-fache Anwendung der Einwegfunktion auf den Wert h_i. Ist das Ergebnis identisch mit dem initialen Commit h_0, so ist h_i ein valides Element der Kette und folglich authentisch (vgl. Definition von Hashfunktionen

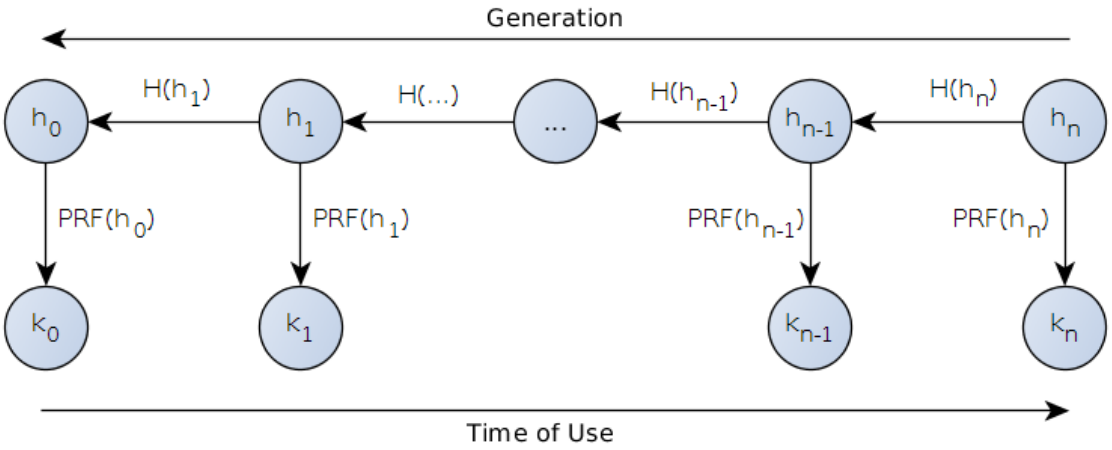


Abbildung 5.6: TESLA's *One-Way Key Chain*. Der Sender berechnet die Kette von Hashwerten anhand eines zufälligen Wert h_n vor und sendet den *Commit*-Wert h_0 an alle potentiellen Empfänger (über einen sicheren Kanal). Aus den Hashwerten h_i werden die eigentlichen Schlüssel k_i durch die Anwendung einer PRF $k_i = PRF(h_i)$ erzeugt. Diese werden zur Berechnung der *TMAC*s von gesendeten Nachrichten eingesetzt, wobei der zuletzt erzeugte Schlüssel zuerst verwendet. Ist das Zeitfenster zur Nutzung eines Schlüssels k_i abgelaufen, so wird dieser nach d Intervallen veröffentlicht und kann von den Empfängern anhand des initialen *Commits* h_0 authentifiziert werden.

in Kapitel 2.3.2.5). Ist dem Empfänger ein anderes Element der Kette h_j (mit $j < i$) bekannt, so genügt die Überprüfung von der Gleichung 5.2, um die Authentizität des neuen Werts sicherzustellen, da jedes Element der Kette als *Commit* für alle weiteren Elemente dient.

$$H^{i-j}(h_i) \stackrel{?}{=} h_j \tag{5.2}$$

Die Anzahl der Kettenelemente, welche der Sender für die Erzeugung der Key Chain vorberechnet hat, zusammen mit der Intervalllänge, beschreibt ein klar definiertes Zeitfenster. Sobald die TESLA-geschützte Kommunikation beginnt, beschrieben durch T_0, kann die Key Chain nur eine beschränkte Zeitspanne verwendet werden, bevor alle Elemente der Kette veröffentlicht wurden. Diese maximale Zeitspanne lässt sich leicht berechnen: Sei n die Anzahl der Kettenelement, so ist die maximale Zeitspanne T_{max} zum Einsatz einer Key Chain gegeben durch $T_{max} = n \cdot I_{int}$.
Je kürzer die Intervalllänge, desto mehr Kettenelemente muss der Sender folglich vorberechnen, um dieselbe Zeitspanne abzudecken. Angenommen eine Kommunikation soll für $10s$ abgesichert werden und die Intervalllänge beträgt $100ms$, so muss der Sender $\frac{10s}{100ms} = 100$ Elemente vorberechnen. Beträgt die Intervalllänge dagegen $2,5ms$, so müsste der Sender für dieselbe Zeitspanne bereits 4000 Elemente vorberechnen.

5.5.3.2 TESLA-Frame

Ein TESLA-Sender überführt Nachrichten für eine gesicherte Übertragung in die folgenden Form:

$$\langle m, \quad i, \quad h_{i-d}, \quad TMAC_{k_i}(\cdot)\rangle \tag{5.3}$$

m Die übertragenen Nachricht.

i Index des Intervalls, in welchem sich der Sender zum Zeitpunkt der Übertragung befindet. Diese Information wird benötigt um Empfängern die Prüfung der TESLA-Sicherheitsbedingung zu ermöglichen.

h_{i-d} Das zuletzt veröffentlichte Element der Kette, erlaubt implizit die Berechnung des Schlüssels k_{i-d} und damit die Authentifizierung aller gesendeten Nachrichten, aus Intervallen $i' \leq i - d$. Dabei definiert der Parameter d die sogenannte *Key Disclosure* Verzögerungszeit, also die Anzahl der Intervalle zwischen Nutzung und Veröffentlichung eines Kettenelements.

$TMAC_{k_i}(\cdot)$ Berechnung der MAC mit dem aktuell verwendeten Schlüssel k_i über das Tripel (m, i, h_{i-d}).

Dabei muss das veröffentlichte Kettenelement h_{i-d} nicht zwangsläufig in jeder übertragenen Nachricht enthalten sein, theoretisch genügt die Übermittlung des Elements einmalig innerhalb eines gegebenen Intervalls. Denkbar ist auch die Aufteilung des Elements auf mehrere Nachrichten um den Overhead zu verringern. Im Sinne der Robustheit des Verfahrens ist jedoch eine Übertragung des vollen Kettenelements in jeder Nachricht von Vorteil, da so selbst im Falle von Paketverlusten eine Authentifizierung früherer Nachrichten möglich ist, sobald eine einzelne Nachricht empfangen wird.
Ein interessanter Aspekt der Konstruktion ist die allgemeine Robustheit der Kette. Ein Empfänger kann einmal empfangene Nachrichten garantiert authentifizieren, solange er zu einem späteren Zeitpunkt, nach der Veröffentlichung des benötigten Schlüssels, ein beliebiges Kettenelement empfängt. Mit Kenntnis des Index, können alle in der Zwischenzeit verlorenen Kettenelement, und damit die verwendeten Schlüssel, wiederhergestellt werden und so gespeicherte Nachrichten nachträglich authentifiziert werden, solange die TESLA-Sicherheitsbedingung erfüllt ist.

5.5.3.3 TESLA-Sicherheitbedingung

Die zentrale Sicherheitsgarantie von TESLA basiert auf der Unumkehrbarkeit von Einwegfunktionen: Da ein Angreifer nicht in der Lage ist die Funktion H umzukehren, kann er aus dem öffentlich bekannten Commit kein Kettenelement berechnen, welches noch nicht veröffentlicht wurde. Folglich ist er nicht in der Lage den aktuell verwendeten Schlüssel des Senders zu bestimmen.
Sobald ein Kettenelement h_i jedoch veröffentlicht ist, kann der Angreifer offensichtlich

beliebige TESLA-Frames unter Verwendung des Schlüssel k_i erstellen, da dieser aus h_i erzeugt wird. Um einen solchen, trivialen Angriff zu erkennen, prüfen Empfänger anhand der Sende- und Empfangszeit einer Nachricht, die TESLA-Sicherheitsbedingung. Dies bedingt eine grobe Zeitsynchronisation zwischen Sender und Empfänger (siehe Abschnitt 5.5.3.5). Entscheidend ist, dass der Empfänger eine Obergrenze für die aktuelle Zeit des Senders kennt, inklusive eventueller Fehler des verwendeten Zeitsynchronisationsverfahrens.
Sei T_r die Zeit, zu welcher ein Empfänger die TESLA-Nachricht $\langle m, i, h_{i-d}, \mathit{TMAC}_{k_i}(\cdot)\rangle$ empfängt und T_s die Obergrenze der Senderzeit. Anhand dieser Informationen kann der Empfänger mittels Gleichung 5.4 das maximale Intervall I_x bestimmen, in welchem sich der Sender aktuell befinden kann.

$$I_x = \left\lfloor \frac{T_s - T_0}{T_{int}} \right\rfloor \tag{5.4}$$

Anhand von I_x und des Systemparameters d kann die TESLA-Sicherheitsbedingung 5.5 geprüft werden.

$$I_x \overset{?}{<} I_i + d \tag{5.5}$$

Informell formuliert fordert die TESLA-Sicherheitsbedingung, dass der Sender sich zum Zeitpunkt des Nachrichtenempfangs bei Empfänger noch nicht in einem Intervall befinden darf, zu welchem der verwendete Schlüssel veröffentlicht ist.
Ist die Sicherheitsbedingung für eine empfangene Nachricht erfüllt, so speichert der Empfänger die Nachricht, bis das Kettenelement h_i durch den Sender veröffentlicht wird. Nach Erhalt des Elements prüft der Empfänger h_i auf dessen Authentizität. Ist die Prüfung erfolgreich, so kann k_i berechnet und mit diesem die gespeicherte Nachricht abschließend authentifiziert werden, indem die zugehörige *TMAC* validiert wird. Unter der Annahme, dass der Sender den Schlüssel k_i bzw. das Kettenelement h_i bis zur Veröffentlichung geheim gehalten hat, garantiert dies die Sicherheitseigenschaft der Senderauthentifizierung.

5.5.3.4 Bootstrapping

Damit ein Empfänger TESLA-Nachrichten authentifizieren kann, benötigt dieser (neben der Zeitsynchronisation mit dem Sender) die initialen (engl. Bootstrap) Informationen. Diese Informationen müssen an alle Empfänger und über einen gesicherten Kanal übertragen werden (einmalig bei Systemstart und erneut, falls der Sender die Key Chain wechselt). Die Bedingung des sicheren Kanals für die Übermittlung der Bootstrap Informationen ist notwendig, um eine manipulation des initialen Commit durch einen aktiven Angreifer zu verhindern. Gelingt die Manipulation des Commit, so kann dieser im späteren Verlauf eigene Schlüssel veröffentlichen und somit scheinbar authentische Nachrichten übertragen.
Die folgenden Informationen werden von Empfängern benötigt, um TESLA-Nachrichten erfolgreich authentifizieren zu können:

T_0 Ein absoluter Zeitpunkt, zu welchem das erste Zeitintervall beginnt. Dieser Wert wird zur Überprüfung der Sicherheitseigenschaft 5.5 benötigt.

T_{int} Die Länge eines Zeitintervalls wird ebenfalls für die Prüfung der Sicherheitseigenschaft 5.5 benötigt.

d Der Systemparameter d beschreibt zusammen mit T_{int} die maximale Verzögerungszeit und damit die Zeitspanne, für welche eine Nachricht gespeichert werden muss. Zudem wird d für die Prüfung der Sicherheitseigenschaft 5.5 benötigt.

h_0 Der initiale Commit der Key Chain des Senders wird benötigt um empfangene Kettenelemente h_i zu authentifizieren.

5.5.3.5 Zeitsynchronisation

Um die zuvor beschriebene Obergrenze für die Senderzeit zu bestimmen, wird eine lose Zeitsynchronisation zwischen Sender und Empfänger benötigt. Dabei ist die Genauigkeit weniger entscheidend als die Kenntnis einer Obergrenze des Fehlers. Die Synchronisation kann mithilfe einer einfachen RTT-Messung, einer gemeinsamen externen Zeitreferenz (z. B. GPS-Zeit) oder bereits synchronisierter Uhren (z. B. mithilfe eines Zeitsynchronisationsprotokolls wie NTP oder PTP) erfolgen.
Die Synchronisation mittels einer externen Zeitreferenz ist von Vorteil, da kein zusätzlicher (extra abgesicherter) Nachrichtenaustausch zwischen Sender und Empfänger (zur Ermittlung der Zeitdifferenz) erfolgen muss. Der Sender übermittelt zu diesem Zweck lediglich in regelmäßigen Abständen eine Schätzung über seine aktuelle Abweichung $\Delta_s + |\epsilon_s|$ gegenüber der Referenzzeit (mit $|\epsilon_s|$ als maximalen Fehler dieser Schätzung). Jeder Empfänger besitzt eine eigene Schätzung $\Delta_r + |\epsilon_r|$ über die Abweichung der lokalen Uhr gegenüber der Referenzzeit. Daraus ergibt sich direkt eine implizite Zeitsynchronisation zwischen Sender und Empfänger $\Delta = \Delta_s + \Delta_r + |\epsilon_s| + |\epsilon_r|$ mit einem maximalen Fehler von $\epsilon = |\epsilon_s| + |\epsilon_r|$.
Für das betrachtete System wird angenommen, dass beide Systeme mit der GPS-Zeit synchronisiert sind, entweder direkt über einen GPS-Empfänger oder durch ein entsprechend genaues Zeitsynchronisationsprotokoll. Für die obige Berechnung bedeutet dies, dass Δ_r und Δ_s im Rahmen der GPS Synchronisationsgenauigkeit liegen. Diese liegt mit hohen Wahrscheinlichkeit unterhalb einer Mikrosekunde - mit 66% Wahrscheinlichkeit sogar bei unter 50ns [88][89]. Der Gesamtfehler Δ wird im Folgenden, sehr konservativ, mit maximal 2ms angenommen.

5.5.3.6 Key Disclosure Delay

Die Wahl des Systemparameters d ist entscheidend für die Authentifikationsverzögerung des TESLA-Systems. Ein zu klein gewähltes d führt zu häufiger Verletzung des Sicherheitskriteriums 5.5, während ein zu großes d zu unnötigen Verzögerungen und

einer erheblichen Anzahl gespeicherter Nachrichten führt. Perring et al. geben in ihrer Arbeit eine Formel für den Mindestwert von d, in Abhängigkeit des maximalen Synchronisationsfehlers ϵ und einer Obergrenze für die maximale Netzwerkverzögerung D_{SR} an [81]:

$$d = \left\lceil \frac{D_{SR} + \epsilon}{T_{int}} \right\rceil + 1 \tag{5.6}$$

Für das betrachtete System können die folgenden Werte angenommen werden:

ϵ Im vorherigen Abschnitt wurde das maximale ϵ auf 2ms eingeschränkt.

D_{SR} Die maximale Netzwerkverzögerung ist im Vergleich zu ϵ gering, da das betrachtete System eine flache Hierarchie (wenige Switches) und eine geringe räumliche Verteilung besitzt. Die Verzögerung liegt im Bereich weniger Mikrosekunden und ist damit mindestens ein Größenordnungen kleiner als das gewählte ϵ.

Für $T_{int} \geq D_{SR} + \epsilon$ ergibt sich $d = 2$. Bei einer konservativen Wahl von $T_{int} = 2,5ms$ liegt die maximale Verzögerung einer Nachricht bei 5ms. Der Wert liegt im Fall von Paketverluste höher, sollte der zur Authentifizierung benötigte Schlüssel erst zu einem späteren Zeitpunkt empfangen werden.
Die Verzögerung kann durch eine optimistischere Abschätzung von ϵ für ein tatsächliches System mit hinreichend genauer Zeitauflösung weiter reduziert werden. Ein zusätzlicher Faktor für die Fehlerabschätzung ist zudem die Genauigkeit, mit welcher das zugrundeliegende System den Empfangszeitpunkt einer Nachricht messen und speichern kann.
Für das betrachtete System ist diese Verzögerung jedoch bereits hinreichend gering, sodass alle relevanten Anwendungsfälle unterstützt werden können, da *adaptive AUTOSAR* die Anwendung auf harte Echtzeitanwendungen explizit ausschließt [18].

5.5.3.7 Senderseitiges Buffering

Perring et al. beschreiben eine Variation von TESLA, in welcher der Sender (anstatt dem Empfänger) Nachrichten für d Intervalle speichert [81]. High-Level Idee des Ansatzes ist eine implizite Authentifizierung von Nachrichten mithilfe von Hashwerten, welche in zuvor gesendeten und authentifizierten Nachrichten enthalten sind.
Seien p_i und p_j zwei Nachrichten, wobei p_j mindestens d Intervalle nach p_i übertragen werden soll. Beide Nachrichten besitzen den klassischen TESLA-Aufbau, jedoch enthält p_i zusätzlich den Hashwert $H(p_j)$ der Nachricht p_j. Dies ist möglich, indem p_i vom Sender so lange zurück gehalten wird, bis p_j bereit ist. p_j wird daraufhin ebenfalls verzögert, um die nachfolgende Nachricht zu authentifizieren.
Kann ein Empfänger das Paket p_i authentifizieren, so kann dieser, anhand von $H(p_j)$, die Nachricht p_j sofort nach dem Erhalt authentifizieren. Die Robustheit von TESLA bleibt dabei erhalten, da die prinzipielle TESLA-Authentifizierung eines Pakets weiterhin möglich ist. Insbesondere bedeutet dies, dass p_j auch dann von einem Empfänger authentifiziert werden kann, wenn p_i nicht empfangen wurde. Allerdings muss der Empfänger

in diesem Fall die Nachricht p_j wie im unmodifizierten Verfahren d Intervalle speichern, bis der zugehörige Intervallschlüssel k_j vom Sender offengelegt wird.

Diese Variante des TESLA-Protokolls erlaubt eine Entlastung der Empfänger im Durchschnittsfall, da weniger Nachrichten empfängerseitig gespeichert werden müssen. Der große Nachteil ist dagegen das schlechtere *worst-case* Verhalten gegenüber dem originalen Verfahren, da bei einem Paketverlust die Authentifizierungsverzögerung auf $2d$ ansteigt. Die Verdopplung entsteht durch die zweimalige Speicherung der Nachricht durch Sender und Empfänger, jeweils für d Zeitintervalle. Senderseitige Pufferung erfolgt enstprechend des modifizierten Verfahrens, während die empfängerseitige Pufferung aufgrund des Paketverlustes der zuvor übertragenen Nachricht zurückzuführen ist.
Zu beachten ist, dass alle Nachricht, welche in den Intervallen I_0 und I_1 gesendet werden, nicht sofort authentifiziert werden können, und folglich der doppelten Zeitverzögerung $2d$ unterliegen.
Generell ist die senderseitige Speicherung der Nachrichten von Vorteil, wenn Empfänger erheblich größeren Einschränkungen bezüglich des vorhandenen Speicherplatzes unterliegen als der Sender.

Die Autoren argumentieren zudem, dass dieses Verfahren robuster gegenüber DoS-Attacken auf den Puffer der Empfänger ist als das ursprüngliche Verfahren [81]. Grundsätzlich ist ein Angriff auf TESLA-Empfänger denkbar, indem ein Angreifer eine hohe Anzahl beliebiger Nachrichten an den Sender übermittelt. Da dieser die empfangenen Nachrichten erst nach d Intervallen authentifizieren kann, müssen alle temporär gespeichert werden. Ist der Puffer des Empfängers zu klein, so kann eine hohe Zahl an empfangenen Nachrichten diesen zum Überlaufen zu bringen. Je nach Implementierung des Puffers, ist so z. B. eine Verdrängung von validen Nachrichten eines Senders durch die invaliden des Angreifers möglich.

5.5.3.8 Zusätzliche Sicherheitseigenschaften

TESLA kann unter bestimmten Bedingungen zusätzlich zur Senderauthentifizierung auch die Sicherheitseigenschaften Freshness und Nonrepudiation zusichern.

Freshness Replay Angriffe auf TESLA-authentifizierte Nachrichten sind grundsätzlich nur innerhalb von d Zeitintervallen nach der ursprünglichen Übertragung möglich. Dies ist der Fall, da nach d Intervallen eine Nachricht aufgrund der TESLA-Sicherheitsbedingung nicht länger von den Empfängern akzeptiert wird. Folglich genügt die Nutzung einer verhältnismäßig kurzen Sequenznummer, um die Freshness übertragener Nachrichten abzusichern.
Für die Beispielwerte $T_{int} = 2,5ms \cdot d = 2 \rightarrow 5ms$ ist bereits ein einfacher 16-Bit

Zähler ausreichend, selbst im Falle eines voll ausgelasteten $2,5$ GBit/s Netzwerks (vgl. Berechnung 5.7).

$$\begin{aligned} \frac{2.500.000.000\ \textit{GBit pro s}}{8 \cdot 64^{\ddagger}} &\leq 4882813\ \textit{Pakete pro s} \\ \frac{4882812}{1000} &\leq 4883\ \textit{Pakete pro ms} \\ 4883 \cdot 5\textit{ms} = 24415 &\ll 2^{16} - 1 = 65535 \end{aligned} \tag{5.7}$$

Der Einsatz eines expliziten Zählers ist nicht notwendig, falls die gesicherten Nachrichten bereits einen hinreichen langen Freshness-Wert enthält.

Nonrepudiation Einer der Vorzüge von digitalen Signaturen ist, dass deren Anwendung die Verbindlichkeitseigenschaft erfüllt. Für TESLA ist dies nicht der Fall, da eine valide wirkende Nachricht von jedem erzeugt werden kann, sobald der zugehörige Schlüssel veröffentlicht wurde.
Durch den Einsatz eines vertrauenswürdigen Time-Stamp Servers (TSS), welcher als passiver Empfänger agiert, kann diese Eigenschaft jedoch auch für TESLA erreicht werden [86]. Der TSS zeichnet alle empfangenen Nachrichten auf und speichert diese zusammen mit dem exakten Empfangszeitpunkt sicher ab. Der dafür benötigte Speicherplatz kann verringert werden, indem der TSS lediglich alle Nachrichten speichert, deren Sicherheitskriterien erfüllt sind. Solange der TSS ehrlich agiert, kann zu einem beliebigen Zeitpunkt die Authentizität und Verbindlichkeit einer Nachricht anhand der TSS-Aufzeichnungen geprüft werden.

5.5.3.9 Analyse und sekundäre Sicherheitseigenschaften

Das TESLA-Verfahren ermöglicht Senderauthentifizierung in einem *1:n* Kontext unter Anwendung einfacher symmetrischer MACs und Hashfunktionen. Zusätzlich kann TESLA unter bestimmten Voraussetzungen auch Freshness und Nonrepudiation sicherstellen. Für die Wahrung der Geheimhaltungseigenschaft muss ein zusätzlicher Gruppenschlüssel verwendet werden, da der Einsatz von TESLA-Schlüsseln die Geheimhaltung nur für d Zeitintervalle sichern kann. Der Overhead pro Paket liegt bei lediglich einer MAC, einem Hashwert und dem Kettenindex. Das Protokoll ist dabei sehr robust gegen Paketverluste und kann durch geeignete Wahl der Parameter (d, T_{int}) an unterschiedliche Szenarien und Systeme angepasst werden.

Der Trade-Off des Verfahrens ist die verzögerte Authentifizierung von Nachrichten, die Notwendigkeit zur senderseitigen Vorberechnung der Key Chain und die sender- bzw.

‡Minimale Länge eines Ethernet Frame. Für eine exakte Berechnung müsste zusätzlich noch die MAC-Preamble und der Inter-Frame-Gap berücksichtigt werden, wodurch die tatsächliche Anzahl übertragener Pakete pro Sekunde geringer ausfällt. Für eine Abschätzung genügt der verwendete Wert von 64 Byte.

empfängerseitige Speicherung von Nachrichten, bis zur Veröffentlichung des Schlüssels durch den Sender. Die Zeitverzögerung der TESLA-Authentifikation hängt dabei stark von den Eigenschaften des betrachteten Netzwerks und der Größe des Zeitsynchronisationsfehlers ab. Für das betrachtete Netzwerk sind beide Parameter niedrig, weshalb eine Zeitverzögerung von etwa 5ms als großzügiger Schätzwert veranschlagt werden kann. Eine solch geringe Verzögerung sollte für alle Anwendungen ohne harte Echtzeitforderung ausreichend gering sein und kann je nach verwendeter Hardware erheblich weiter verringert werden.
TESLA fordert von Empfängern eine lose Zeitsynchronisation mit dem Sender und einen gesicherten Kanal zur sicheren Übertragung der initialen Bootstrap-Informationen. Beide Voraussetzungen sind für die Sicherheit des Verfahrens von entscheidender Bedeutung, weshalb diese als sekundäre Sicherheitsattribute des Systems zu betrachten sind. Die gesicherte Übertragung der Bootstrap-Informationen kann mithilfe sicherer Unicast-Kommunikation erfolgen. Die Absicherung der losen Zeitsynchronisation ist eine größere Herausforderung. Wird ein Zeitsynchronisationsprotokoll (z. B. NTP oder PTP) verwendet, so muss dieses Protokoll ebenfalls vor Manipulation geschützt sein. Wird stattdessen eine indirekte Synchronisation gegenüber einer gemeinsamen Zeitbasis (z. B. GPS) verwendet, muss das System eine mögliche Manipulation dieser Zeitbasis verhindern - oder zumindest erkennen können. Durch die weite Verbreitung von GPS, existiert eine breite Wissensbasis für GPS-Spoofing, welche das Aussenden gefälschter GPS-Signale mithilfe einer Software gesteuerten Funkantenne erlaubt [90]. Ein Angreifer kann auf diesem Wege die interne Zeit des Systems verändern, um erfolgreiche Replay-Angriffe auf TESLA-geschützte Nachrichten durchzuführen. Um einen solchen Angriff erkennen zu können, kann die absoluten GPS-Zeit an eine lokale und garantiert monotone Zeitreferenz gebunden werden (z. B. ein einfacher Zähler), um Zeitsprünge in ein früheres Intervall entdecken zu können.
Mit der Variante für senderseitige Pufferung existiert eine TESLA-Variante zur Entlastung schwächerer Empfänger, wobei sich das worst-case Zeitverhalten des Verfahrens verschlechtert. Die angesprochene höhere Robustheit gegenüber DoS-Attacken auf den Nachrichtenspeicher des Empfängers kann auch mithilfe einer Prüfung der Gruppenauthentizität, bei Einsatz einer zusätzlichen symmetrischen MAC erfolgen.
Die Notwendigkeit zur Vorberechnung der Hashelemente durch den Sender vor Kommunikationsbeginn stellt mit Hinblick auf die strengen Vorgaben für den Systemstart eine Herausforderung dar. Über die Laufzeit des Systems hinweg betrachtet ist der Overhead gering (lediglich eine Hashoperation pro Zeitintervall), jedoch ist (abhängig von der Kettenlänge und Intervalldauer) eine hohe Zahl von Hashwerten für die längerfristige Absicherung der Kommunikation notwendig. Lösungsansätze für dieses Problem werden im Kapitel 6.3.2.1 besprochen.

Insgesamt erscheint das TESLA-Verfahren als gut geeignet für den Einsatz in dem betrachteten System. Die Anwendung auf das betrachtete System wird im nächsten Kapitel 6 genauer untersucht. Besonderes Augenmerk liegt dabei auf der Absicherung des Service Discovery Protokolls und der sicheren Übermittlung der Bootstrap-Informationen.

5.5.4 BiBa One-Time Signature

Das BiBa-Verfahren (für *Bins and Balls*) basiert auf der Sicherheitseigenschaft, dass es für einen Angreifer schwierig ist eine *k-fache* Kollision einer Einwegfunktion zu finden [91]. Für die Erzeugung einer Nachrichtensignatur setzt das Verfahren n sogenannter SEALs (engl. SElf Authenticating vaLues) ein - vorberechnete Zufallswerte, welche mithilfe eines öffentlichen Schlüssels des Senders von jedem Empfänger authentifiziert werden können. Dies wird durch eine *One-Way SEAL Chain* Konstruktion erreicht, analog zu der bereits vorgestellten *Key Chain* des TESLA-Verfahrens, wobei die Werte der Key Chain den aktuell verwendeten SEALs entsprechen. Durch wiederholte Anwendung der entsprechenden Hashfunktion auf empfangene SEALs können diese folglich direkt vom Empfänger authentifiziert werden.
Zum Signieren einer Nachricht berechnet der Sender für alle aktuell verwendeten SEALs $s_{i,t}$ mit $i \in \{1..n\}$ deren Hashwert $G_h(s_{i,t})$ unter Verwendung einer Hashfunktion aus einer Familie von Hasfunktionen G_h. Die Wahl der jeweiligen Hashfunktion wird dabei durch den Hashwert $F(m) = h$ der zu signierenden Nachricht bestimmt. Für die BiBa-Signatur einer Nachricht wird eine *k-fache* Kollision $G_h(s_{i,t}) = ... = G_h(s_{j,t})$ mit $s_{i,t} \neq ... \neq s_{j,t}$ gesucht. Ist eine solche Kollision gefunden, kann der Sender seine Nachricht übermitteln. Als zusätzliche Authentifizierungsinformation, hängt der Sender die kollidierenden SEALs $s_{i,t}$, ..., $s_{j,t}$ an die Nachricht an. Der Sender kann die Authentizität der SEALs durch Prüfung der SEAL-Chain-Eigenschaft bestimmen. Ist diese erfolgt, kann die *k-fach* Kollision der gegebenen SEALs für die empfangene Nachricht - und damit deren Authentizität - geprüft werden.
Die Parameter k und n des Verfahrens werden so gewählt, dass der Sender eine hohe Wahrscheinlichkeit hat, eine *k-fache* Kollision innerhalb von zwei Versuchen zu entdecken. Die Sicherheit des Verfahrens basiert dabei darauf, dass der Sender viele SEALs besitzt, während ein Angreifer lediglich diejenigen SEALs kennt, die zuvor (als Teil einer Signatur) veröffentlicht wurden. Aufgrund der geringen Zahl öffentlich gewordener SEALs, ist es sehr unwahrscheinlich, dass ein Angreifer für eine beliebige Nachricht eine *k-fache* Kollision der ihm bekannten SEALs finden kann. Da diese Wahrscheinlichkeit mit der Zahl veröffentlichter SEALs stetig steigt, muss der Sender in regelmäßigen Abständen den Satz der verwendeten SEALs wechseln. Dies erfolgt analog zum TESLA-Verfahren durch die Einteilung in Zeitintervalle, nach deren verstreichen das nächste Element der Kette verwendet wird. Folglich müssen Sender und Empfänger ebenfalls lose zeit-synchronisiert sein.

5.5.4.1 Analyse und Anforderungen

Das BiBa-Verfahren garantiert Senderauthentifizierung unter Einsatz von einfachen Hashoperationen, wobei auf den Sender eine höhere Rechenlast entfällt als auf die Empfänger, da der Sender n-Hashfunktionen berechnen muss gegenüber k-Berechnungen für jeden Empfänger. Der inhärente Vorteil des BiBa-Verfahren im Vergleich zu TESLA ist

die sofortige Authentifizierung der BiBa-Signaturen, welche das Puffern von Nachrichten unnötig macht. Die Robustheit gegenüber Paketverlusten und der Einsatz effizienter Operationen teilen beide Verfahren.
Analog zu TESLA benötigt das BiBa-Verfahren ebenfalls eine lose Zeitsynchronisation, sowie die Übermittlung der Bootstrap-Informationen (Intervalllänge, Startzeit und die initialen *SEAL-Commits*) über einen gesicherten Kanal (vgl. Abschnitt 5.5.3.4). Ebenfalls gemein ist beiden Verfahren die Notwendigkeit zur Vorberechnung der verwendeten Key bzw. SEAL Chains durch den Sender. Im Vergleich zu TESLA sind jedoch die *SEAL-Commits* sehr groß und stellen somit eine echte Schwierigkeit für den Einsatz des Verfahrens dar. Je nach Systemparametern und verwendeten Hashfunktionen, sind die initialen *Commits* in einer Größenordnung von 10kByte und größer [91]. Die Signaturlänge beträgt dabei - ebenfalls Parameterabhängig - 128 Bytes und mehr. Für einen Einsatzzweck in einem aktuellen System sind beide zuvor genannten Werte um den Faktor 2 zu niedrig anzusehen, da Perring in der Arbeit den Einsatz der Hashfunktion MD5 zugrunde legt. MD5-Hashwerte haben eine Länge von 128-Bit, wobei das Verfahren als gebrochen und unsicher gilt. Ein aktuell als sicher betrachtetes sicheres Hashverfahren (z. B. SHA-256) besitzt dagegen eine Hashlänge von mindestens 256-Bit. Bereits die 128 Byte Signaturen sind mit einem Blick auf den SOME/IP Standard jedoch zu groß, um diese ohne Änderung des Standards zur Absicherung einzusetzen, da dieser lediglich 56 Bytes für Erweiterungen vorsieht.

6 Sichere SOME/IP Kommunikation

Im vorherigen Kapitel wurden eine Reihe von Sicherheitsmechanismen vorgestellt und deren grundsätzliche Tauglichkeit für das betrachtete System, sowie der entsprechenden Anforderungen und Sicherheitseigenschaften diskutiert. Im Folgenden soll ein ganzheitliches Konzept zur Absicherung der SOME/IP Kommunikation unter Verwendung einer Kombination der vorgestellten Verfahren erstellt werden. Die eingesetzten Verfahren werden im Anschluss auf deren Kompatibilität mit dem SOME/IP-Standard überprüft und die Sicherheit des Konzepts informell untersucht.

6.1 Gesamtkonzept

Die SOME/IP Kommunikation lässt sich grob in die Anwendungsfälle und davon direkt abgeleitete Funktionalitäten aufteilen, welche bereits in Kapitel 4.2.2 beschrieben wurden. Tabelle 6.1 zeigt die Zuordnung der im vorherigen Kapitel beschriebenen Sicherheitsverfahren zu den Anwendungsfällen und den zugrundeliegenden Kommunikationstypen. Sämtliche Unicast-Kommunikation wird mithilfe von (D)TLS abgesichert, welches in der Lage ist, alle gewünschten Sicherheitseigenschaften (Authentizität, Geheimhaltung und Freshness) sicherzustellen. Für Multicast-Situationen wird zur Senderauthentifizierung das TESLA-Verfahren eingesetzt, eine kurze Sequenznummer stellt die Freshness sicher, während ein einfacher symmetrischer Gruppenschlüssel zur Wahrung der Geheimhaltungseigenschaft zum Einsatz kommt. Dieser Schlüssel kann, je nach Anwendungsszenario, mithilfe eines Schlüsselbaums verwaltet werden, um eine dynamische Gruppenmitgliedschaft effizient und sicher zu implementieren. Die Übertragung des Gruppenschlüssels und eventuell benötigter Bootstrap-Informationen an die Gruppenmitglieder erfolgt mithilfe von (D)TLS-abgesicherter Unicast-Kommunikation. Wo möglich kann diese Übertragung an bereits vorhandene Nachrichten angehängt werden, z. B. an die Bestätigung zu einem erfolgten Gruppenbeitritt, um den Kommunikationsoverhead gering zu halten. Die SOME/IP-SD Phase erfordert einen neuen Ansatz, da die Möglichkeiten des SOME/IP-Standards nicht ausreichen, um eine sichere Gruppenkommunikation für die Service Discovery zu erstellen. Der Ansatz verwendet einen zentralen SOME/IP-Master Knoten, welcher eine sichere Gruppe erstellt und (D)TLS als Zugangskontrolle einsetzt. Dieses Konzept wird in Kapitel 6.3.3 vorgestellt und analysiert.

Anwendungsfall	Typ	Verfahren	Sicherheitseigenschaften
RPC & Felder Gruppenbeitritt (Events & SD)	Unicast	(D)TLS	Authentizität, Geheimhaltung, Freshness
Eventgruppe: Benachrichtigung	Multicast (*1:n*)	TESLA	(Sender-)Authentizität
		Sequenznummer	Freshness
		Gruppenschlüssel, Schlüsselbaum	Geheimhaltung
Service Discovery	Multicast (*m:n*)	(D)TLS & TESLA	(Sender-)Authentizität
		Sequenznummer	Freshness
		Gruppenschlüssel, Schlüsselbaum	Geheimhaltung

Tabelle 6.1: Zuordnung von Sicherheitsverfahren zu SOME/IP Anwendungsfällen und garantierte Sicherheitseigenschaften.

6.2 Unicast-Absicherung: (D)TLS

Zur Absicherung der Unicast-Kommunikation zwischen zwei Kommunikationspartner kommt gemäß dem Gesamtkonzept (D)TLS zum Einsatz. Dabei kann die laufende Kommunikation mit nur einer einzelnen AEAD-Operation je Nachricht abgesichert werden. Auf modernen CPUs stehen dazu zumeist spezielle Befehlssatzerweiterungen zur Verfügung, z. B. Intels *AES New Instruction Set*(AES-NI), welche die Performanz der AEAD-Operation erheblich beschleunigen.
Im Bezug auf Performanz ist folglich insbesondere der (D)TLS-Handschlag zu untersuchen, da hierbei mehrere asymmetrische Operation durchgeführt werden müssen, um eine server- und clientseitige Authentifikation zu erreichen. Da asymmetrische Operation sehr rechenintensiv sind, muss die Performanz im Folgenden genauer betrachtet werden, um die strikten Anforderungen an die Startzeit des Systems zu beachten.

6.2.1 Performanz des (D)TLS Handschlags

Für einen vollen (D)TLS Handschlag fallen eine Reihe kryptographischer Operationen an, wobei die Anzahl der durchgeführten asymmetrischen Operation für beide Parteien identisch ist, solange wechselseitige Authentifikation erreicht werden soll.

- Nach dem Erhalt des Zertifikats des Kommunikationspartners prüft der Empfänger zunächst die Gültigkeit des empfangenen Zertifikat anhand eines gespeicherten Herstellerzertifikats, bzw. einer entsprechenden Zertifikatskette. Je nach Länge

der zu prüfenden Kette müssen beide Parteien zumindest eine asymmetrische *Verify*-Operation durchführen.

- Für den Beweis, dass der Absender im Besitz des privaten Schlüssels ist, welcher zu dem zuvor übermittelten Zertifikats gehört, signiert die jeweilige Partei das Transscript (der bisherigen Nachrichten) mit dem privaten Schlüssel. Dieser Schritt ist notwendig, um eventuelle Replay-Attacken durch den Angreifer zu verhindern. Es fallen für jede Partei dabei jeweils eine asymmetrische *Sign*- und *Verify*-Operation an.

Den praktischen Einfluss von TLS auf die Performanz des *IoT*-Systems *HiveMQ* ist Teil eines aufwendigen Benchmarks der Firma dc-sqaure [92]. HiveMQ nutzt ein einfaches Publish-Subscribe-Konzept, um Nachrichten von IoT-Clients an einen zentralen, leistungsstarken Server zu übertragen und Nachrichten anderer Clients über diesen Weg zu empfangen. Der Benchmark untersucht den Overhead, welcher für den zentralen Knoten entsteht, falls alle Verbindungen mit TLS abgesichert werden.
Abbildung 6.1 zeigt den direkten Vergleich zwischen der mittels TLS abgesicherten Implementierung (grüner Graph) und der ungesicherten Variante (in rot). Während der ersten 10 Minuten des Versuchs bauen die Clients (jeweils in Gruppen zu je 100) eine Verbindung zum Server auf und ab Minute 14 senden alle Clients eine Nachricht alle 10 Sekunden. Für die Authentifizierung werden 4096-Bit RSA-basierte Zertifikate und die (noch) aktuelle Protokollversion 1.2 eingesetzt.
Deutlich ist bei Betrachtung der grünen Kurve die außerordentliche Belastung des Servers zu erkennen, welche eindeutig auf die Durchführung des TLS-Handschlags zurückzuführen ist.
Ebenso deutlich ist die Tatsache zu erkennen, dass die TLS-gesicherte Kommunikationen nach Abschluss des Handschlags einen nur minimalen Overhead im Vergleich zur ungesicherten Verbindung aufweist.

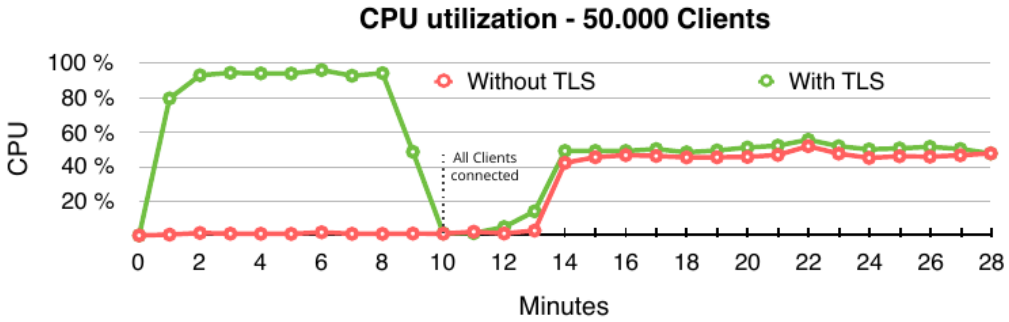


Abbildung 6.1: Vergleich der Belastung eines IoT-Servers mit und ohne den Einsatz von TLS zur Absicherung der Kommunikation mit den Clientsystemen [92].

Im Benchmark wird für den Verbindungsaufbau der vollständige TLS (1.2) Handschlag eingesetzt, mit beidseitiger Authentizitätsprüfung mittels Zertifikaten. Neben dem kompletten Handschlag existieren zusätzlich die *1-RTT* und *0-RTT* Varianten, welche einen zuvor ausgehandelten symmetrischen Schlüssel zum Wiederaufbau der Verbindung einsetzen (siehe Abschnitt 5.3.2.3). Beide Varianten besitzen erhebliche Geschwindigkeitsvorteile, da alle asymmetrische Operationen entfallen. Zugleich wird die Größe der ausgetauschten Handschlag-Informationen erheblich verringert, da keine Zertifikate ausgetauscht werden.
Für zeitkritische Situationen oder schwache Systeme stellt *Session Resumption* eine ausgezeichnete Möglichkeit zum Aufbau sicherer Verbindungen dar, ohne den Einsatz rechenintensiver asymmetrischer Operationen. Der initiale, volle TLS-Handschlag kann in regelmäßigen Intervallen wiederholt werden, um die zugrundeliegende Sicherheitsgarantien zu erneuern. Diese Auffrischung kann somit zu einem Zeitpunkt durchgeführt werden, zu denen das System ausreichend Zeit, Rechenkapazität und Energie zur Verfügung hat.

6.2.2 (D)TLS Kompatibilität zu SOME/IP

Aufgrund der geringen Länge des (D)TLS Headers (siehe Abschnitt 5.3.2.4) kann das Verfahren bedenkenlos standardkonforme SOME/IP-Frames kapseln. Je nach eingesetzter TLS-Variante verbleiben $56 - 24 = 32$ Bytes bei der Verwendung von TLS bzw. $56 - 32 = 24$ Bytes für zusätzliche Erweiterungen.
Eine Möglichkeit zur Steigerung der Sicherheit, ist der Einsatz von *Record Padding*, einer TLS-Erweiterung [74]. Mit dieser Erweiterung kann an eine übertragene Nachricht eine zufällige Anzahl 0-Bytes angehängt werden, bevor diese der AEAD-Operation unterzogen wird. Der Einsatz und die maximale Länge des Padding kann von den Kommunikationspartnern während des TLS-Handschlags ausgehandelt werden, wobei die Maximallänge einer Nachricht immer eine obere Grenze darstellt. Dies ist hilfreich um zu verhindern, dass ein Angreifer Aufschluss über den Umfang der Kommunikation zweier Partner durch Analyse der Längen von ausgetauschten Nachrichten erlangt.

Konzeptionell betrachtet kann (D)TLS als zusätzliche Schicht zwischen SOME/IP und dem UDP/IP bzw. TCP/IP Kommunikationsstack des Systems platziert werden. Jede Nachricht, welche physisch an eine andere ECU übertragen bzw. von dieser empfangen wird, muss die (D)TLS Schicht passieren und wird entsprechend dem Protokoll verarbeitet. Im Fall einer Verbindung zu einem Ziel, zu welchem noch keine sicherer (D)TLS-Kanal besteht, wird zunächst ein entsprechender Handschlag ausgeführt.

6.3 Multicast-Absicherung: TESLA

Entsprechend dem vorgestellten Konzept wird TESLA für die Garantie der Senderauthentifizierung in Multicast-Szenarien eingesetzt. Für die Sicherheit von TESLA ist, neben der Zeitsynchronisation der Teilnehmer, vor allem die sichere Übertragung der initialen Bootstrap-Informationen (im Folgenden abgekürzt mit BS_{sender}) in beiden Szenarien sicherzustellen. Zunächst muss jedoch die Kompatibilität von TESLA mit dem SOME/IP Standard diskutiert werden.

6.3.1 TESLA Kompatibilität zu SOME/IP

In Kapitel 5.5.3.2 ist der Aufbau eines TESLA-Frames beschrieben. Für die Konformität mit dem SOME/IP Standard ist insbesondere die Gesamtlänge der TESLA-Zusatzinformationen (Intervallindex i, Kettenelement h_{i-d}) und die Größe der TMAC von Bedeutung. Für eine direkte Kompatibilität mit dem Standard sollte die Gesamtlänge der Headerinformationen weniger als 56 Bytes betragen.

Intervallindex Die Länge dieses einfachen Zählers ist beliebig und kann entsprechend der Key Chain Länge gewählt werden. Einzige Bedingung für den Index ist, dass jedes Kettenelement ein eindeutig i besitzen muss. Läuft der Zähler über, können Nachrichten aus den nachfolgenden Intervallen nicht länger authentifiziert werden, da diese die TESLA-Sicherheitsbedingung verletzen. Wird ein kurzer Zähler verwendet so muss in regelmäßigen Abständen ein Wechsel der Key Chain erfolgen. Bei einer angenommenen Intervalllänge von $2,5ms$, läuft bereits ein 16-Bit Zähler nach etwa zweieinhalb Minuten über. Wird eine sehr lange Kette verwendet, kann ein 24-Bit Zählers eingesetzt werden, welcher (für die angenommen $2,5ms$) eine Laufzeit von $11,5$ Stunden erlaubt.

Kettenelement Die Größe dieser Zusatzinformation ist äquivalent zur Länge eines Hashwertes der verwendeten Funktion. Die Unumkehrbarkeit der Funktion ist Teil der zentralen Sicherheitsgarantie des Verfahrens, weshalb eine als kryptographisch sicher eingestufte Hashfunktion eingesetzt werden muss. Entsprechend der BSI-Richtlinien für kryptographische Verfahren [35] kommen dafür aktuell alle Mitglieder der *SHA-2* und *SHA-3* Familie infrage, welche eine Hashwertlänge von mindestens 256-Bit aufweisen. Die Länge des übertragenen h_{i-d} Elements beträgt folglich mindestens 32 Bytes. Die Auswahl des tatsächlichen Verfahrens kann abhängig von der verwendeten Hardware und Software erfolgen, um bereits implementierte Methoden wiederzuverwenden, oder eine vorhandene Hardwareunterstützung auszunutzen. Aufgrund der Lebenszeit des betrachteten Systems sollte zudem eine modularisierte Implementierung gewählt werden, welche den Austausch der verwendeten Hashfunktion erlaubt (siehe Kapitel 2.3.2.2).

TMAC Die Länge des $TMAC_{k_i}(\cdot)$ Elements wird durch das verwendete MAC-Verfahren definiert. Zu Erwarten ist hierbei der Einsatz eines standardisierten AES-CMAC-Verfahrens und folglich eine Länge von 16 Bytes.

Insgesamt fügt TESLA folglich mindestens 50 Byte an zusätzlichen Informationen zu jeder SOME/IP-Nachricht hinzu. Dies liegt noch im Rahmen der verfügbaren 56 Byte, welche der Standard für Erweiterungen vorsieht. Soll dagegen ein Hashverfahren eingesetzt werden, dessen Länge größer als 32 Bytes ist, so muss die Übertragung von h_{i-d} auf Kosten der Robustheit über mehrere Nachrichten verteilt werden.
Je nach verwendeter Netzwerkhardware kann die maximale Nachrichtenlänge jedoch auch (erheblich) größer als 1500 Bytes ausfallen (sog. *Jumbo-Frames*), wodurch zusätzliche Bytes für Sicherheitserweiterungen zur Verfügung stehen. Dies ist offensichtlich nur möglich, falls alle beteiligten Systeme *Jumbo-Frames* unterstützen.

6.3.1.1 Aufbau eines TESLA-authentifizierten Frames

Abbildung 6.2 zeigt den möglichen Aufbau eines TESLA-authentifizierten Frames. Die Länge des Intervallindex und Sequenzzählers kann je nach Wahl von T_{int} variiert werden, wobei die Maximallänge des TESLA-Headers 56 Byte nicht übersteigen sollte, um die Kompatibilität mit dem SOME/IP Standard nicht zu verletzen.
Der reservierte Bereich kann beispielsweise für einen zusätzlichen (möglicherweise gekürzten) MAC genutzt werden, welche mithilfe eines symmetrischen Gruppenschlüssels das sofortige Prüfen der Gruppenauthentizität ermöglicht. Dies verhindert triviale DoS-Angriffe auf den Pufferspeicher von Empfängern durch einen externen Angreifer (vgl. Diskussion in Abschnitt 5.5.3.7).
Optional kann ein kurzer Zähler (oder ein einfaches Bit-Flag) verwendet werden, welcher die ID der aktuellen TESLA-Key Chain bezeichnet. Dies steigert die Robustheit des Verfahrens, im Fall eines Kettenwechsels. Verpasst ein Empfänger die Nachricht bezüglich des Wechsels auf eine neue Kette, kann die Nutzung einer solchen ID Empfänger auf den verpassten Wechsel aufmerksam machen.

Für den weiteren Verlauf der Arbeit wird die Notation 6.1 verwendet, um einen solchen TESLA-Frame zu beschreiben. Dazu wird die TESLA-Instanz des *sender* mit den jeweils aktuellen Werte für das Intervall I_i verwendet.

$$TESLA_{sender}\left\langle \cdot \right\rangle \tag{6.1}$$

An eine gegebene Nachricht (alle Elemente innerhalb der spitzen Klammern) wird der TESLA-Header angefügt, welcher zumindest die Informationen I_i und h_{i-d} sowie einen hinreichend langen Freshness-Wert enthält. Über die konkatenierte Nachricht wird schließlich der *TMAC* berechnet und ebenfalls angefügt. Dies erlaubt einem Empfänger die Nachricht gemäß dem TESLA-Verfahren zu authentifizieren, solange er in Besitz der notwendigen Bootstrap-Informationen BS_{sender} ist.

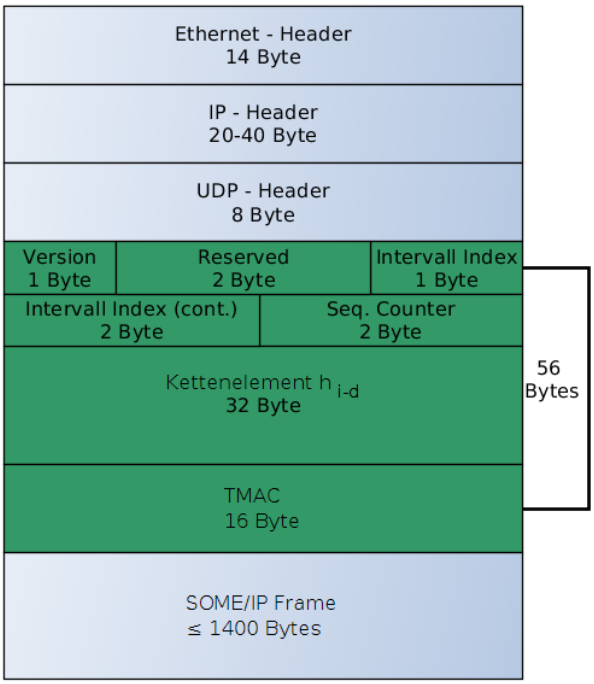


Abbildung 6.2: Möglicher Aufbau eines TESLA-authentifizierten SOME/IP Frames.

6.3.2 TESLA-gesicherte Multicast-Gruppen

Für den Aufbau einer TESLA-geschützten Multicast-Gruppe muss ein Sender im wesentlichen zwei Schritte durchführen. Zunächst muss er die zu verwendende Key Chain vorberechnen und im Anschluss die Bootstrap-Informationen über einen sicheren Kanal an alle Empfänger verteilen. Gemäß der Übersichtstabelle 6.1 verwenden TESLA-Gruppen zur Wahrung der Geheimhaltung zusätzlich einen symmetrischen Gruppenschlüssel. Anstelle eines einfachen Gruppenschlüssels kann stattdessen auch ein Schlüsselbaum zum Einsatz kommen (vgl. Kapitel 5.4.2.3).
Unabhängig von der angewendeten Variante wird im Folgenden angenommen, dass alle notwendigen Informationen zusammen mit den Bootstrap-Informationen an die Empfänger gesendet werden. Gleiches gilt dabei für die notwendigen Information über verwendete Verfahren und Länge der einzelnen Headerinformationen.
Während dem laufenden Betrieb einer TESLA-Gruppe muss der Sender zudem darauf achten, in regelmäßigen Abständen seinen geschätzten Synchronisationsfehler zur externen Zeitreferenz zu veröffentlichen. Dies ist notwendig, um Empfängern die Abschätzung der aktuellen Senderzeit zu ermöglichen, welche zur Prüfung der Sicherheitseigenschaft verwendet wird.
Zusätzlich muss der Sender sicherstellen, dass in jedem Zeitintervall zumindest eine Nachricht gesendet wird, welche das aktuell veröffentlichte Kettenelement h_{i-d} enthält.

Dies ist notwendig damit Empfänger zuvor übertragene Nachrichten authentifizieren können.

6.3.2.1 Vorberechnung der Key Chain

Eine zentrale Eigenschaft von TESLA ist die inhärente Kopplung der Key Chain an die Zeitintervalle. Um die Authentifizierungsverzögerung gering zu halten, sollten die Intervalle so kurz wie möglich gewählt werden. Dies bedeutet jedoch aufgrund der Kopplung von Kette und Intervallen, dass eine hohe Anzahl an Kettenelementen vorberechnet werden muss, auch wenn nur wenige Kettenelemente tatsächlich zum Einsatz kommen. Die Anzahl der zu berechnenden Elemente lässt sich leicht anhand der folgenden Formel bestimmen:

$$\textit{Kettenlänge} = \frac{\textit{Laufzeit}}{Intervalllänge} \tag{6.2}$$

Diese Eigenschaft ist auf den angenommenen Anwendungsfall von TESLA (konstanter Nachrichtenstrom) zurückzuführen. Im betrachteten System dagegen ist es durchaus wahrscheinlich, dass Nachrichten deutlich seltener übertragen werden als einmal pro Intervall. Als Beispiel für dieses Problem kann die SOME/IP-SD Phase betrachtet werden. Nach SOME/IP Standard ist zu erwarten, dass ein Sender während der Service Discovery Phase nur wenige Nachrichten sendet. Eine mögliche Beispielkonfiguration für die SOME/IP-SD *Repetition*-Phase sieht eine Dauer von $40\,ms$ bei vier Wiederholungen vor. Bei einer Intervalllänge von $2,5\,ms$ müsste ein Sender folglich mindestens $\frac{40}{2,5} = 16$ Kettenelemente vorberechnen - obwohl nur vier Nachrichten übertragen und authentifiziert werden. Die tatsächliche Länge der dazu verwendeten Kette liegt dabei realistisch betrachtet noch deutlich höher, da das Startintervall T_0 und der Start der Wiederholungsphase nicht direkt zusammenfallen und ein gewisse Reserve für die Authentifizierung weiterer Multicast-Nachrichten nach Abschluss der Service Discovery benötigt wird.
Noch deutlicher wird der Unterschied im Fall von *Update-on-Change* Eventgruppen - z. B. ein einfacher Sensor zur Messung der Drehzahl. Steht das Fahrzeug für 10 Sekunden im Stillstand (z. B. an einer Ampel), wird der Sensor keine Updates versenden, da sich der Wert (0) nicht ändert. Dennoch laufen die Intervalle der zugehörige TESLA-Instanz weiter ab - für das gegebene Beispiel insgesamt $\frac{10s}{2,5ms} = 4000$ Intervalle, ohne dass ein einzelnes Element verwendet wird.

Ein naiver Ansatz zum Entkoppeln von Key Chain und Intervallen (unter Beibehaltung des Sicherheitskriteriums und eigenem Index für Kettenelemente) ist durch einen Angreifer mit vollständiger Netzwerkkontrolle leicht auszuhebeln. Dazu muss dieser lediglich alle gesendeten Nachrichten für d Zeitintervalle verzögern oder zurückhalten, um so das offengelegte Kettenelement zu erhalten. Anschließend kann der Angreifer beliebige Nachrichten erzeugen, solange er weitere Übertragungen des Senders blockiert und anhand des erhaltenen Schlüssels (scheinbar) valide Nachrichten erzeugt.

Die Vorberechnung einer erheblichen Zahl von Kettenelementen ist folglich für den Einsatz von TESLA unumgänglich. Um diesen nicht-unerheblichen Rechenaufwand praktikabel zu halten, bieten sich im wesentlichen zwei verschiedene Ansätze an.

Gespeicherte Ketten Anstatt die Kette während dem Systemstart frisch zu erzeugen, kann diese asynchron (z. B. parallel zum normalen Betrieb) oder offline vorberechnet und in einem sicheren Speicher für den späteren Gebrauch abgespeichert werden. Zu beachten ist, dass bei steigender Länge der Kette auch die Menge der zu speichernden Elemente steigt. Da die gespeicherte Kette in einem sicheren und persistenten Speicher abgelegt werden muss, kann dies ein erheblich limitierender Faktor sein. Ein Trade-Off zwischen Speicherplatz und Berechnung einzelner Werte ist dabei möglich, z. B. in dem nur jedes $\log(n)$-te Element der Kette gespeichert wird. Für den Zugriff auf ein Kettenelement müssen dann maximal $\log(n)$ Hashoperationen durchgeführt werden [81].

Kurze Ketten Effiziente Implementierungen und mögliche Hardwareunterstützung erlauben die Berechnung mehrerer hunderttausend Hashoperationen pro Sekunde, selbst auf weniger leistungsstarken Systemen. Die im MinnowBoard (vorgestellt in Abschnitt 2.1.2.1) verbaute CPU wurde in einem Bitcoin-Mining Benchmark auf dessen Performanz im Hinblick auf die Berechnung von Hashfunktionen untersucht und erreichte eine Geschwindigkeit von $1,83$ Millionen Hashberechnungen pro Sekunde [93]. Selbst bei einer vollständigen Ausnutzung der verfügbaren Rechenleistung für die Vorberechnung der Kette (eine unrealistische Annahme), könnte innerhalb einer Sekunde dennoch nur eine Kette mit einer maximalen Laufzeit von etwa 12 Sekunden berechnet werden.*

In den wenigen Sekunden des Startvorgangs kann folglich keine ausreichend lange Kette vorberechnet werden, welche für die gesamte Laufzeit des Systems ausreichend ist. Stattdessen kann lediglich eine kurze Kette vorberechnet werden, welche für den Startvorgang und die ersten Betriebssekunden ausreicht. Während diese Kette zur Authentifizierung eingesetzt wird, kann die nächste Kette im Hintergrund vorberechnet werden. Sobald die initiale Kette erschöpft ist, kann auf die nächste gewechselt werden. Nachteil dieses Verfahrens ist der steigende Overhead für den häufigen Austausch erneuerter Bootstrap-Informationen.

Denkbar sind auch Kombinationen beider Ansätze, z. B. der Einsatz gespeicherter Ketten für den Systemstart und der Wechsel auf parallel berechnete Ketten nach den ersten Betriebssekunden.
Zudem ist zu bemerken, dass prinzipiell eine TESLA Key Chain je ECU ausreichend ist und nicht jede Multicast-Gruppe eine eigene Kette benötigt. Da bereit während des Systemstarts TESLA zur Absicherung der SOME/IP-SD Protokolls zum Einsatz kommt (siehe Abschnitt 6.3.3), muss bei der Erstellung einer neuen Gruppe in der

* $\frac{1800000\ Kettenelemente}{2{,}5ms\ Intervalldauer} = 720000ms\ Laufzeit = 12\ Minuten.$

Regel keine neue Kette berechnet oder neue Bootstrap-Informationen ausgetauscht werden. Stattdessen kann die schon bestehende TESLA-Instanz zur Absicherung der Gruppenkommunikation eingesetzt werden.
Für eine Ende-zu-Ende Absicherung der Kommunikation auf Service-Ebene kann eine zusätzliche TESLA-Instanz verwendet werden, um einen spezifischen Service anstelle einer ECU als TESLA-Sender zu authentifizieren. Dies kann für kritische Services (z. B. OTA-Updates) zur weiteren Steigerung der Sicherheitsstufe eingesetzt werden, jedoch genügt prinzipiell eine eindeutige Authentifizierung der absendenden ECU, vorausgesetzt das System erfüllt die Trusted-Plattform Anforderung.

6.3.2.2 SOME/IP Eventgruppe

Der erste Fall für SOME/IP Multicast-Kommunikation stellen Eventgruppen dar. Diese verwenden einfache *1:n* Multicast-Nachrichten und sind folglich direkt mit TESLA kompatibel. Der gesicherte Austausch der Bootstrap-Informationen ist in dieser Situation einfach möglich, indem das standardisierte Publish-Subscribe-Protokoll von SOME/IP verwendet wird (siehe Abbildung 6.3).
Für den Beitritt in einer Gruppe sendet ein Client (im Beispiel der ADAS-Service) die *subscribeEventgroup* (Nachricht 6.3) direkt an den anbietenden Server (im Beispiel der adSensor-Service). Ist der Client zum Beitritt in die Gruppe berechtigt, antwortet der Server mit der *subscribeEvent-groupAcknowledge*-Nachricht. Diese kann gemäß SOME/IP Standard einen *Configuration*-Eintrag enthalten, mit welcher der Server beliebige Konfigurationsdaten an den Client übermitteln kann (vgl. Kapitel 3). Mithilfe eines solchen Eintrags können die Bootstrap-Informationen $BS_{adSesnsor}$ direkt in der Antwortnachricht (6.4) übermittelt werden. Geht diese verloren, so wird der betreffende Clients nach einem Timeout eine erneute *subscribeEventgroup*-Nachricht senden, welche vom Server entsprechend beantwortet wird.
Eventnachrichten werden durch den Server stets TESLA-authentifiziert und mit dem Gruppenschlüssel verschlüsselt übertragen (vgl. Nachricht 6.5).

$$\Pi_c \rightarrow \Pi_s : \left\{ subscribeEventgroup \right\}^{s}_{k_{sk}} \tag{6.3}$$

$$\Pi_s \rightarrow \Pi_c : \left\{ ACK, k_{gk}, BS_s \right\}^{s}_{k_{sk}} \tag{6.4}$$

$$\Pi_s \rightarrow \Pi_c : TESLA_s \left\langle \{ eventData \}^{s}_{k_{gk}} \right\rangle \tag{6.5}$$

Der Ansatz besitzt praktisch keinen Overhead für das System, da die *subscribeEventgroupAcknowledge*-Nachricht gemäß dem SOME/IP Standard in jedem Fall gesendet werden muss. Lediglich eine Änderung der Bootstrap-Informationen, z. B. eine Änderung des Gruppenschlüssels oder beim Wechsel auf eine neue Key Chain, erzeugt einen Kommunikationsoverhead, da eine neue Nachricht für deren Übertragung erstellt werden muss. Für eine effiziente Übermittlung der neuen Daten sollte ebenfalls Multicast-Kommunikation

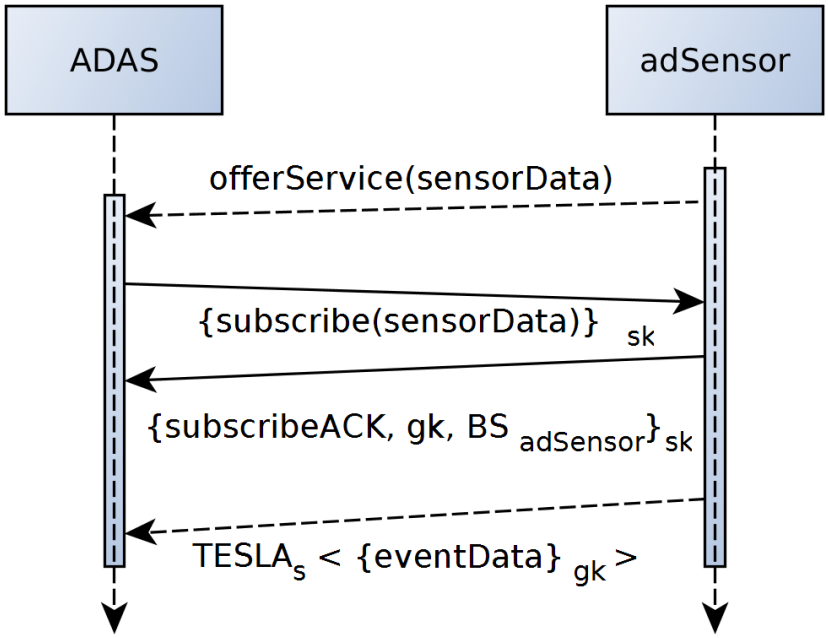


Abbildung 6.3: Abonnieren einer TESLA-geschützten Eventgruppe unter Verwendung eines zuvor ausgetauschten symmetrischen Sitzungsschlüssels *sk*.

zur Anwendung kommen. Da SOME/IP, nach Abschluss der Service Discovery, keine weitere Übertragung von Konfigurationsinformationen vorsieht, ist dafür ein zusätzliches Protokoll notwendig.

Soll ein solcher Wechsel vollständig mit Mitteln des SOME/IP Standards erreicht werden, so kann die Eventgruppe durch den Server einfach neu gestartet werden. Dadurch werden alle interessierten Clients zu einer Wiederholung des Gruppenbeitritts gezwungen, nach dessen Abschluss sie die neuen Informationen erhalten.
Für einen Neustart muss der Server lediglich eine neue *offerService*-SOME/IP-SD Nachricht senden, welche über einen aktualisierten Eintrag verfügt. Der Eintrag muss dazu dieselbe Eventgruppen-ID aufweisen, jedoch in zumindest einem anderen Feld verschieden sein. Dies lässt sich beispielsweise durch ein *Epoch*-Eintrag erreichen, analog zu dessen Einsatz in DTLS (siehe Abschnitt 5.3.2.2). Dadurch wird die Nachricht von den Empfängern als Aktualisierung der Service Informationen interpretiert und die alte Gruppe gilt als aufgelöst. Alle Clients, welche die Eventgruppe auch weiterhin abonnieren wollen, müssen anschließend einen neuen *subscribeEventgroup*-Handschlag durchführen. Dieser Ansatz erfordert den Austausch von $2n$ Unicast-Nachrichten pro Empfänger. Eine Änderung der Bootstrap-Informationen tritt in dynamischen Gruppen und bei der Verwendung von kurzen TESLA-Key Chains häufig auf, wodurch dieser Ansatz einen nicht zu vernachlässigbaren Overhead erzeugt.

Die effizientere Lösung für das Problem ist der Einsatz eines einfachen Protokolls zur Übertragung der benötigten Informationen, analog zu den sogenannten *Post-Handshake*-Nachrichten des (D)TLS Protokolls. Diese werden eingesetzt, um einen Schlüsselwechsel zu einem beliebigen Zeitpunkt des Verfahrens oder eine erneute Authentifizierung der Parteien durchzuführen. Nachrichten eines solchen Typs werden lediglich von der entsprechenden Schicht interpretiert und nicht an höhere Schichten weitergegeben.
Bei der Verwendung eines solchen Protokolls muss der Absender lediglich eine verschlüsselte und mit der altern TESLA-Instanz authentifizierte Multicast-Nachricht mit den neuen Informationen senden. Für den Wechsel des symmetrischen Gruppenschlüssels, z. B. bei Austritt eines Gruppenmitglieds, kann die *reKey*-Nachricht der *Leave*-Methode des verwendeten Schlüsselbaums als Teil der Multicast-Nachricht übertragen werden (siehe Kapitel 5.4.2.3).

6.3.2.3 Informelle Sicherheitsanalyse

Die Sicherheit des beschriebenen Ansatzes basiert vollständig auf den Sicherheitsgarantien der zugrundeliegenden Unicast-Verbindung zwischen Client und Server, da über diese die Bootstrap-Informationen des Senders übertragen werden. Gemäß dem Konzept aus Abschnitt 6.1 wird zur Absicherung einer solchen Verbindung (D)TLS eingesetzt. Da (D)TLS alle notwendigen Sicherheitseigenschaften für die übertragenen Daten garantiert, wodurch ein gesicherter Kanal für die Übertragung der Bootstrap-Informationen bereitsteht.
Folglich sind Empfänger in der Lage, mit TESLA-authentifizierte und dem Gruppenschlüssel verschlüsselte, Nachrichten des Senders zu authentifizieren und entschlüsseln. Durch den Einsatz von TESLA ist die Senderauthentifizierung für Eventdaten gegeben, weshalb selbst ein Angreifer mit vollständiger ECU-Kontrolle über einen der Empfänger nicht in der Lage ist Eventdaten zu fälschen, oder den eigentlichen Sender zu imitieren.

6.3.3 Absicherung des SOME/IP-SD Protokolls

Für den Einsatz von TESLA zur Absicherung der Kommunikation in einem *m:n* Szenario ist es notwendig, dass jeder Sender eine eigene TESLA-Instanz besitzt und alle potentiellen Empfänger die Bootstrap-Informationen aller Sender über einen sicheren Kanal erhalten. Das Problem ist analog zu einem Schlüsselaustausch in einem *m:n* Kontext (vgl. Kapitel 5.4.1).
Die High-Level Idee zur Lösung des Problems ist der Einsatz eines *Service Discovery Master* (SDM) als vertrauenswürdige Drittpartei (siehe Abbildung 6.5). Ein solcher SDM kann als zusätzlicher Service in das Beispielsystem aus Kapitel 3.1.1 eingefügt werden.

SDM Service zum Aufbau und der Verwaltung einer sicheren SOME/IP-SD Gruppe. Die angebotene *sdControl*-Eventgruppe dient zum effizienten und sicheren Austausch

der Bootstrap-Informationen mit dem SDM und deren anschließende Verteilung an alle Berechtigten. Zusätzlich lassen sich mithilfe dieser Gruppe Veränderungen in der Gruppenstruktur (z. B. durch Beitritt bzw. Austritt eines Mitglieds oder Schlüsselwechsel) effizient übertragen.
Alle Services[2], welche an der gesicherte SOME/IP-SD Kommunikation teilnehmen möchten, müssen dieser Gruppe beitreten.

Abbildung 6.4 und Tabelle 6.2 zeigen die veränderten Abhängigkeiten der Services bei Einsatz eines SDM.

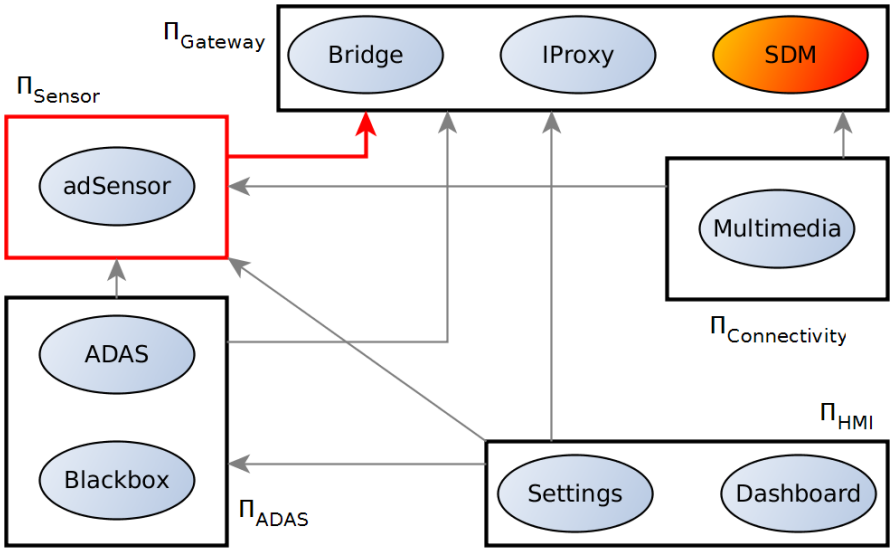


Abbildung 6.4: Abhängigkeiten des Beispielsystems nach dem Einfügen des SDM-Service, wodurch die rot markierte Abhängigkeitsbeziehung hinzukommt.

ID	Service	Methoden	Eventgruppen	Benötigt	Knoten
1	ADAS	config	warnings	transmit, sensorData, CANread, CANwrite, sdControl	Π_{ADAS}
2	adSensor	transmit	sensorData	sdControl	Π_{Sensor}
3	Bridge	CANwrite	CANread	sdControl	$\Pi_{Gateway}$
4	IProxy	iRequest	-	sdControl	$\Pi_{Gateway}$
5	Multimedia	-	-	sensorData, iRequest, sdControl	$\Pi_{Connectivity}$
6	Settings	-	-	config, CANwrite, sdControl	Π_{HMI}

[2]Genau betrachtet besitzt jede ECU einen SOME/IP-SD Prozess, der die Service Discovery für diese ECU implementiert. Alle Services der ECU sind von der korrekten Funktion dieses Prozesses abhängig, wobei dieser wiederum den SDM für eine sichere Kommunikation voraussetzt. Somit ergibt sich eine indirekte Abhängigkeit aller Services des Systems gegenüber dem SDM-Service.

7	Dashboard	-	-	warnings, CANread, sensorData, sdControl	Π_{HMI}
8	Blackbox	logMe	-	alle Eventgruppen, iRequest, sdControl	Π_{HMI}
9	SDM	-	sdControl	-	$\Pi_{Gateway}$

Tabelle 6.2: Übersicht über die Liste der SOME/IP Services des Beispielsystems, erweitert um den SDM-Service zur Verwaltung einer gesicherten SOME/IP-SD Gruppe.

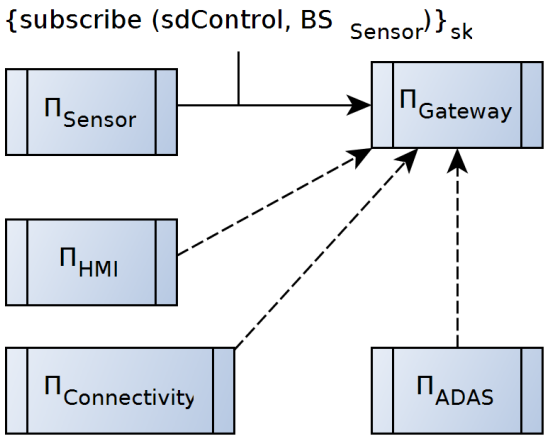
Der Beitritt eines potentiellen Teilnehmers Π_i erfolgt dabei in zwei zusätzlichen Schritten, welche vor dem Start des eigentlichen SOME/IP-SD Protokolls ausgeführt werden müssen.

Anmeldephase Während der ersten Phase abonnieren alle potentielle Teilnehmer die *sdControl*-Gruppe des SDM. Die Anmeldung erfolgt (analog zu Kapitel 6.3.2.2) durch den Aufbau einer gesicherten (D)TLS-Verbindung von Π_i zu Π_{SDM}, falls diese noch nicht besteht. Anschließend sendet Π_i seine individuellen Bootstrap-Informationen BS_i als Teil der *subscribeEventgroup*-Nachricht an den SDM (vgl. 6.6 und Teilbild 6.5a).
Ein passiver Teilnehmer des Protokolls kann dabei auf die Übertragung eigener Bootstrap-Informationen verzichten. Ein Beispiel für einen solchen passiven Teilnehmer stellt eine vereinfachte Variante des *Blackbox* Service dar, welcher keinen expliziten SOME/IP Service anbietet.
Die Nachricht 6.6 ist durch den (D)TLS-Sitzungsschlüssel k_{sk} verschlüsselt und authentifiziert. Je nach Autorisationskriterium müssen Teilnehmer zudem noch zusätzliche Informationen übertragen, z. B. ein Attributzertifikat, welches sie zum Beitritt in die Service Discovery Gruppe berechtigt.
Falls der SDM von der Autorisation des Teilnehmers überzeugt ist, sendet er die *subscribeEventgroupAcknowledge*-Nachricht 6.7, welche die Bootstrap-Information des SDM, die Multicast-IP Adresse der sicheren SOME/IP-SD Gruppe und den Gruppenschlüssel k_{SD} enthält (siehe Teilbild 6.5b).

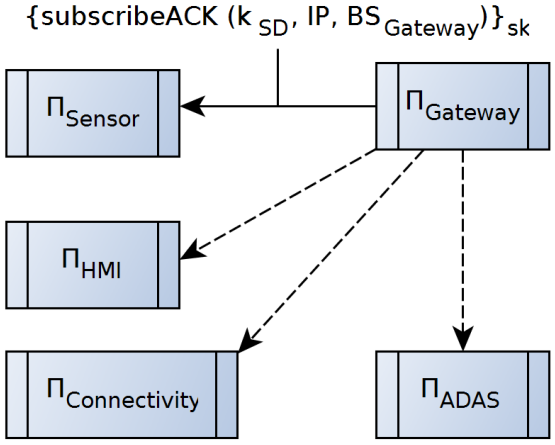
$$\Pi_i \rightarrow \Pi_{SDM} : \left\{ BS_i \right\}^s_{k_{sk}} \tag{6.6}$$

$$\Pi_{SDM} \rightarrow \Pi_i : \left\{ BS_{SDM}, IP, k_{SD} \right\}^s_{k_{sk}} \tag{6.7}$$

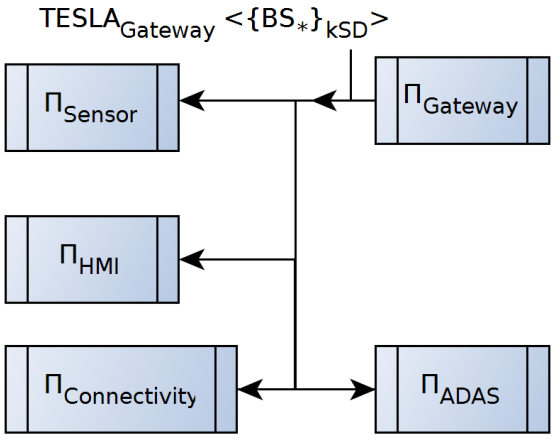
Austauschphase Die zweite Phase umfasst im Wesentlichen die (wiederholte) Übertragung der Nachricht 6.8 durch den SDM an die *sdControl*-Eventgruppe (siehe Teilbild 6.5c). Zentrale Information der Nachricht ist die Liste der Bootstrap-Informationen aller Teilnehmer, welche der SDM während der ersten Phase autorisiert hat. Diese Liste wird mit BS_* beschrieben. Da der verwendete Kanal unzuverlässig ist (und ohnehin in jedem TESLA-Intervall das aktuelle Kettenelement h_{i-d} veröffentlicht



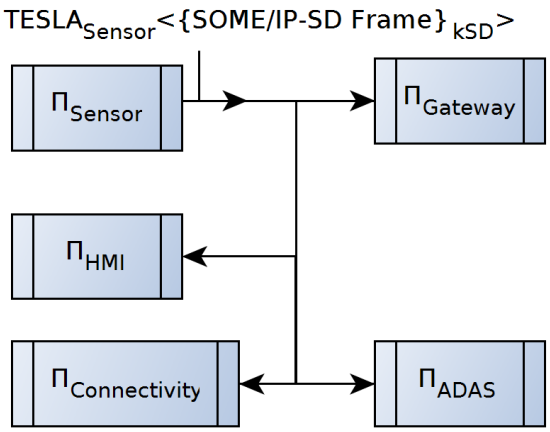
(a) Alle ECUs treten der sicheren *sdControl* Gruppe des SDM bei und übertragen die eigenen Bootstrap-Informationen über eine gesicherte Unicast-Verbindung.



(b) Alle berechtigten ECUs erhalten eine Beitrittsbestätigung, welche die Bootstrap-Informationen des SDM, die Multicast-Adresse *IP* und den Gruppenschlüssel k_{SD} enthält.



(c) Nach Beitritt aller ECUs sendet der SDM ein verschlüsseltes und senderauthentifiziertes *sdControl*-Event an alle Clients, welches die gesammelten Bootstrap-Informationen (BS_*) aller legitimen Teilnehmer enthält.



(d) Legitime Gruppenmitglieder können nun verschlüsselte und senderauthentifizierte SOME/IP-SD Nachrichten an die Multicast-Gruppe senden und von anderen legitimen Mitgliedern empfangen.

Abbildung 6.5: Aufbau einer sicheren SOME/IP-SD Multicast-Gruppe unter Verwendung eines zentralen SDM-Service, welcher auf $\Pi_{Gateway}$ ausgeführt wird.

werden muss) wird diese Nachricht wiederholt gesendet, jeweils mit den aktuellen Werten für I_i, k_i und h_{i-d}. Die Anzahl der Wiederholungen sowie eventuelle Pausen können dabei systemabhängig definiert werden.
Zusätzlich kann die Nachricht einen geteilten Startzeitpunk T_0 enthalten, welcher einen gemeinsamen Startzeitpunkt aller TESLA-Instanzen definiert. Der Einsatz des geteilten T_0 ist dabei optional, jedoch mit Hinblick auf die Performanz von Vorteil. Durch einen gemeinsamen Startzeitpunkt ist auch die TESLA-Sicherheitseigenschaft für alle Parteien identisch (unter der Annahme das T_{int} ebenfalls identisch ist). Werden individuelle Startzeitpunkte verwendet, so müssen Empfänger eine Sicherheitseigenschaft je möglichem Sender verwalten und berechnen können. Ein gemeinsames T_0 ist jedoch bei häufigen Wechsel einer TESLA Key Chain oder dem nachträglichen Beitritt eines Senders problematisch. In solchen Situationen ist der Einsatz individueller Startzeitpunkte je Sender praktikabler.

$$\Pi_{SDM} \rightarrow \textit{SD-Group} : \textit{TESLA}_{SDM} \left\langle \{BS_*, T_0\}^s_{k_{SD}} \right\rangle \tag{6.8}$$

Nach Abschluss der zweiten Phase ist jeder Teilnehmer in der Lage, TESLA-authentifizierte SOME/IP-SD Nachrichten an die Gruppe zu senden und von jedem anderen Teilnehmer zu empfangen, da die benötigten Bootstrap-Informationen aller Teilnehmer bekannt sind (siehe Teilbild 6.5d). Nachricht 6.9 zeigt eine mögliche Übertragung eines SOME/IP-SD Frame (*sdInfo*) durch einen autorisierten Teilnehmer. Durch den zusätzlichen Einsatz des geteilten Gruppenschlüssels kann neben der TESLA-Senderauthentifizierung auch die Geheimhaltungseigenschaft für das SOME/IP-SD Protokolls gewährleistet werden.

$$\Pi_i \rightarrow \textit{SD-Group} : \textit{TESLA}_i \left\langle \{\textit{sdInfo}\}^s_{k_{SD}} \right\rangle \tag{6.9}$$

6.3.3.1 Informelle Sicherheitsanalyse

Die Sicherheit des zuvor beschriebenen Verfahrens basiert im Wesentlichen auf den Garantien von (D)TLS und TESLA. Beide Protokolle werden (im Rahmen der getroffenen Annahmen) als sicher eingestuft, weshalb im Folgenden lediglich eine (informelle) Untersuchung der Kombination beider Verfahren vorgenommen wird. Zusätzlich wird angenommen, dass der Gruppenmeister Π_G ehrlich agiert - also insbesondere nicht von einem Angreifer kontrolliert wird. Solange die Trusted-Plattform Annahme hält, darf ein Angreifer mit Applikationskontrolle auf dem SDM anwesend sein. Lediglich eine vollständige ECU-Kontrolle des SDM kann von dem vorgestellten Verfahren nicht toleriert werden. Diese ist entscheidend, da Π_G alle sicherheitskritischen Operationen ausführt, insbesondere die Wahl des Gruppenschlüssels k_{SD}, die Authentifikationsprüfung von Teilnehmern und die Verteilung der empfangenen Bootstrap-Informationen aller Teilnehmer.

- Der Aufbau einer sicheren (D)TLS-Verbindung zwischen Π_i und Π_{SDM} muss als Teil des ersten Protokollschrittes erfolgreich durchgeführt werden oder bereits zuvor abgeschlossen sein. Dies ist notwendig, um die ausgetauschten Nachrichten der Anmeldephase mittels des ausgehandelten TLS-Sitzungsschlüssels k_{sk} zu authentifizieren und verschlüsseln. Eine Manipulation oder Replay der Nachricht durch den Angreifer ist nur möglich, falls dieser den symmetrischen Sitzungsschlüssel k_{sk} kennt.
- Um in den Besitz des Sitzungsschlüssels zu gelangen, muss der Angreifer entweder das TLS-Protokoll brechen, die Trusted-Plattform Annahme verletzt sein oder er muss vollständige ECU-Kontrolle über den Client besitzen. Die ersten beiden Fälle sind per Annahme ausgeschlossen, während der letzte Fall Teil des *worst-case* Szenarios ist. Eine legitime Verbindung eines manipulierten Clients kann durch Π_{SDM} nicht erkannt werden, wodurch ein Angreifer unter Umständen in die Gruppe aufgenommen wird. Ziel des Systems soll in einem solchen Fall sein, die Fähigkeiten des Angreifers auf legitime Aktionen der übernommen ECU einzuschränken (vgl. Kapitel 4.2.1).
- Die Antwortnachricht (6.7) enthält alle Informationen, welche Π_i benötigt, um TESLA-authentifizierte Nachrichten von Π_{SDM} zu empfangen. Zusätzlich enthält die Nachricht den gewählten Gruppenschlüssel k_{SD} und die Multicast-Adresse für das SOME/IP-SD Protokoll. Die Nachricht wird nur übertragen, falls Π_{SDM} überzeugt ist, dass Π_i zur Teilnahme berechtigt ist. Die exakte Implementierung dieser Zugangskontrolle kann systemabhängig sein - denkbar ist aber der Einsatz von Attributzertifikaten oder lediglich die Forderung nach einem validen, herstellerzertifizierten Zertifikat und damit dem Zustandekommen einer TLS-Verbindung zwischen Π_i und Π_{SDM}.
- Nach Ablauf der Anmeldephase besitzt Π_{SDM} eine Liste der Bootstrap-Informationen aller akzeptierten Teilnehmer. Die zweite Phase des Protokolls besteht darin, diese Liste möglichst effizient und sicher an alle Gruppenmitglieder zu verteilen. Dazu sendet Π_{SDM} die Multicast-Nachricht (6.8) an die zuvor verteilte Gruppenadresse. Die Wahrung der Sicherheitseigenschaften dieser Nachricht ist entscheidend für die Sicherheit des gesamten Verfahrens, weshalb diese im Folgenden genauer betrachtet wird.

  Geheimhaltung und Gruppenauthentizität Beide Eigenschaften lassen sich durch die Anwendung eines AEAD-Verfahrens unter Verwendung des von Π_{SDM} während der Anmeldephase verteilten Gruppenschlüssels k_{SD} sicherstellen. Die sichere Übertragung des Schlüssels ist von der verwendeten (D)TLS-Verbindung garantiert und auf autorisierte Gruppenmitgliedern beschränkt. Da der Schlüssel symmetrisch und allen Gruppenmitgliedern bekannt ist, kann die Anwendung des AEAD-Verfahrens lediglich Geheimhaltung und Gruppenauthentizität garantieren. Diese Eigenschaften genügen, um Angriffe durch einen Angreifer mit Applikation- oder Netzwerkkontrolle zu verhindern. Im

Fall einer vollständigen ECU-Kontrolle können jedoch beide Eigenschaften nicht länger garantiert werden, da der Angreifer den (ihm bekannten) Gruppenschlüssel zur Entschlüsselung und Manipulation übertragener SOME/IP-SD Nachrichten einsetzten kann.
Die Verletzung der Geheimhaltungseigenschaft von SOME/IP-SD Nachrichten hat verhältnismäßig geringe Auswirkungen (vgl. Tabelle 4.5) auf das System, zumal die Wahrscheinlichkeit eines Angriffs aufgrund der getroffenen Maßnahmen erheblich gesunken ist. Anstatt lediglich lesenden Zugriff auf das Netzwerk zu besitzen, muss ein Angreifer eine ECU vollständig kontrollieren um die SOME/IP-SD Daten lesen zu können. Dies erfordert erheblich höhere Systemkenntnisse und Fachwissen als zuvor, wodurch das Safety Level dieses Angriffs in den akzeptablen Bereich fällt.
Die Verletzung der Authentifizierungseigenschaft durch einen Angreifer mit ECU-Kontrolle ist erheblich schwerwiegender, auch wenn das Threat Level, aufgrund der gestiegenen Anforderungen an einen erfolgreichen Angriff, deutlich geringer ist. Für eine weitere Steigerung der Sicherheit muss Senderauthentifizierung der übertragenen Nachrichten erreicht werden.

Senderauthentifizierung Um diese Sicherheitseigenschaft sicherzustellen, nutzt der SDM die eigene TESLA-Instanz und überträgt die gesammelte BS_*-Liste als TESLA-Frame (vgl. Abschnitt 5.5.3.2). Die benötigten Bootstrap-Informationen BS_{SDM} haben alle Empfänger als Teil der TLS-gesicherten Nachricht (6.7) erhalten, wodurch die Bedingung an einen sicheren Kanal zur Übertragung der Bootstrap-Informationen erfüllt ist. Folglich sind alle Gruppenmitglieder in der Lage, die empfangene Nachricht anhand des TESLA-Kriteriums zu authentifizieren und eventuelle Replay-Attacken zuverlässig zu erkennen.
Aufgrund der TESLA-Garantien ist kein Gruppenmitglied außer dem SDM in der Lage diese Nachricht zu erstellen oder erfolgreich zu manipulieren. Insbesondere gilt dies auch im Fall eines Angreifers mit vollständiger ECU-Kontrolle über einen der Gruppenmitglieder.

- Die Sicherheitseigenschaften aller nachfolgend übertragenen SOME/IP-SD Nachrichten (6.9) werden analog erreicht. Ein potentieller Angreifer kann, selbst wenn dieser ein legitimes Gruppenmitglied vollständig kontrolliert, die SOME/IP-SD Nachrichten eines anderen, nicht vom Angreifer kontrollierten, Teilnehmers Π_i nicht manipulieren, solange er die von Π_i verwendete TESLA-Instanz nicht brechen kann.
  Dazu müsste der Angreifer die übertragenen Bootstrap-Informationen des Senders manipulieren, die Zeitsynchronisation der Kommunikationspartner steuern oder Zugriff auf dessen Key Chain besitzen. Der erste Fall wird durch die Sicherheitseigenschaften der entsprechenden Nachrichten (6.6 und 6.8) zusammen mit der Annahme des vertrauenswürdigen SDM verhindert. Der zweite Fall ist ein wichtiges sekundäres Sicherheitsattribut des TESLA-Verfahrens und muss, ebenso wie der letzte Fall, durch das zugrundeliegende System abgesichert werden.

6.3.3.2 Erweiterung: Dynamische Gruppen

Das Protokoll kann, durch eine minimale Änderung in der ersten Phase, dynamische Mitglieder effizient unterstützen. Dazu erstellt Π_{SDM} einen Schlüsselbaum zu Verwaltung des Gruppenschlüssels (vgl. Kapitel 5.5.2).
Anstatt des Gruppenschlüssels k_{SD} sendet SDM in der Antwortnachricht (6.7) den jeweils eindeutigen Knotenschlüssel kek_i an den Empfänger. Nach Abschluss der Anmeldephase kann SDM einen Schlüsselbaum aufbauen, welcher zumindest so viele Blätter wie angemeldete Gruppenmitglieder besitzt. Als Teil der Austauschphase kann SDM dann einen *reKey*-Block als Teil der Nachricht (6.8) übertragen, welche den gesamten Schlüsselbaum verschlüsselt überträgt. Der *reKey*-Block enthält alle KEKs der inneren Knoten des Baumes, jeweils mit beiden Blattknoten verschlüsselt. Durch iteratives Entschlüsseln der einzelnen KEKs entlang des Pfades vom Blatt zur Wurzel erhalten alle Gruppenmitglieder schließlich den Gruppenschlüssel k_{SD}.
Soll die *backward-secrecy* Eigenschaft bei einem Gruppenbeitritt erhalten bleiben, so kann der SDM vor dem Senden der Antwortnachricht 6.7 an Π_j einen Wechsel des Gruppenschlüssels k_{SD} durchführen. Für das SD-Protokoll ist jedoch anzunehmen, dass *backward-secrecy* nicht notwendig oder sogar unerwünscht ist. Π_j könnte beispielsweise eine Reihe von SOME/IP-SD Nachrichten (und deren Empfangszeitpunkt) aufgezeichnet haben, bevor er den Gruppenbeitritt abgeschlossen hat. Im Anschluss an den Erhalt der benötigten Informationen kann Π_j die zuvor gesendeten Nachrichten authentifizieren, entschlüsseln und verwenden. Wird *backward-secrecy* verwendet, ist dies aufgrund des Schlüsselwechsels nicht möglich, wodurch Π_j alle benötigten Services mithilfe von *findService*-Nachrichten erfragen muss, anstatt auf die bereits gesendeten Nachrichten zurückgreifen zu können.
Handelt es sich bei Π_j um einen rein passiven Teilnehmer, so kann der SDM die BS_*-Liste direkt in Nachricht 6.7 senden und der Beitritt ist abgeschlossen. Überträgt Π_j als Teil der Nachricht 6.6 jedoch seine Bootstrap-Informationen BS_j, so muss die Liste BS_* aktualisiert und an alle Empfänger übertragen werden. Dazu führt der SDM die zweite Phase des Verfahrens erneut aus und überträgt die aktualisierte Liste an alle Teilnehmer, wodurch auch Π_j in Besitz aller anderen Bootstrap-Informationen gelangt.

Verlässt Π_j die Gruppe, oder läuft seine zeitlich beschränkte Autorisation ab, so kann der SDM einen Schlüsselwechsel analog zur *Leave*-Prozedur durchführen (vgl. Abschnitt 5.4.2.3). Zusätzlich überträgt er eine aktualisierte BS_*-Liste, aus welcher die Bootstrap-Informationen des verlassenden Knotens (BS_j) entfernt wurde. Alternativ ist auch eine *revoke*-Nachricht denkbar, in welcher der SDM lediglich den verlassenden Knoten identifiziert und alle Teilnehmer die Bootstrap-Informationen des entsprechenden Knotens ihrer lokalen Liste entfernen. Dies kann hilfreich sein, falls BS_* sehr groß ist.
Der verlassende Teilnehmer kann zwar weiterhin die Nachrichten der verbleibenden Teilnehmer authentifizieren (bis zum nächsten Wechsel der Key Chain), aber aufgrund des veränderten Gruppenschlüssels nicht länger entschlüsseln, wodurch die *forward secrecy* gegenüber Π_j gewahrt bleibt. Da die Bootstrap-Informationen von Π_j ungültig

sind, können Empfänger von Π_j gesendete Nachrichten nicht länger authentifizieren und verwerfen sie.

Um dieses Verfahren auch gegen einen Angreifer mit voller Netzwerkkontrolle und gegen Paketverluste abzusichern, wird ein zusätzliches Sicherheitskriterium benötigt, um sicherzustellen dass ein Gruppenausschluss erfolgt. Erreicht die *revoke*-Nachricht bzw. die aktualisierte BS_*-Liste einen oder mehrere Empfänger nicht, so werden diese auch weiterhin alle von Π_j gesendete Nachrichten akzeptieren, auch nach dessen Gruppenausschluss. Dies kann durch gezieltes Unterdrücken der entsprechenden Nachrichten durch einen Angreifer oder einfache Paketverluste geschehen.
Um einfachen Paketverlusten entgegen zu wirken, kann der SDM die entsprechende Nachricht entweder mehrfach übertragen, Client-Bestätigungen nach Erhalt anfordern oder die Übertragung über einen zuverlässigen Kanal (z. B. eine TCP-Unicast-Verbindung) durchführen. Alle diese Ansätze genügen jedoch nicht, um einen Angreifer mit voller Netzwerkkontrolle an der Unterdrückung sämtlicher Nachrichten des SDM zu verhindern. Stattdessen kann der SDM für die übertragene BS_*-Liste eine maximale Lebenszeit definieren, wobei die regelmäßigen Übertragungen der BS_*-Liste durch den SDM die Lebenszeit erneuern. Nach Ablauf der Lebenszeit müssen die Teilnehmer die Liste zwingend verwerfen, wodurch die Authentifizierung weiterer Nachrichten nicht länger möglich ist. Unterdrückt nun ein Angreifer die *revoke*-Nachricht bzw. die aktualisierte Liste, so ist er lediglich für die verbleibende Lebenszeit der Liste in der Lage authentifizierte Nachrichten zu senden. Die TESLA-Sicherheitsbedingung verhindert das Fälschen der von SDM gesendeten BS_*-Liste. Somit stellt die Lebenszeit der Liste eine harte obere Grenze für die Zeitspanne eines möglichen Angriffs dar.
Ein solcher Ansatz erlaubt triviale DoS-Angriffe durch das Unterdrücken der regelmäßigen Nachrichten des SDM, wodurch die gesicherte Kommunikation abbricht. Ein solcher Angriff ist jedoch durch einen Angreifer mit vollständiger Netzwerkkontrolle stets möglich. Erheblich schwerwiegender wäre dagegen eine Situation, in welcher ein Angreifer durch den SDM aus der Gruppe entfernt werden soll, der Angreifer jedoch die *revoke*-Nachricht unterdrückt und weiterhin als legitimes Gruppenmitglied agieren kann.

6.3.3.3 Erweiterung: Multi-Master Betrieb

Um einen *single-point-of-failure* zu vermeiden, können mehrere SDM Service-Instanzen auf verschiedenen ECUs allokiert werden. Die SDMs wählen nach dem Systemstart dynamisch (oder statisch) eine der Instanzen aus, welcher die Rolle des SOME/IP-SDM übernimmt. Sollte der betreffende Knoten ausfallen, können die verbleibenden Instanzen dessen Rolle übernehmen. Das vorgestellte Verfahren ist dabei lediglich als Ausfallsicherung zu sehen, nicht jedoch zur Steigerung der Sicherheit.
Für das Verfahren wird eine statisch bekannte IP-Multicast-Adresse (IP_{SDM}) vorausgesetzt, welche von allen Clients beobachtet wird.

Alle SDM-Instanzen senden in regelmäßigen Abständen TESLA-gesicherte Heartbeat-Nachrichten (6.10) an diese Adresse. Inhalt einer solchen Nachricht ist die zufällig gewählte (oder statische) ID der Instanz (ID_k), sowie deren IP-Adresse (IP_k).

$$\Pi_{SDM_k} \rightarrow IP_{SDM} : TESLA_{SDM_k} \langle ID_k, IP_k \rangle \tag{6.10}$$

Wird der Heartbeat einer anderen SDM-Instanz Π_{SDM_l} entdeckt, speichert Π_{SDM_k} die Nachricht sowie den Empfangszeitpunkt und baut eine TLS-Verbindung zu dieser Instanz auf. Handelt es sich bei Π_{SDM_l} um einen autorisierten SDM-Knoten (dies kann anhand des TLS-Handschlags geprüft werden), so tauschen die Parteien ihre jeweiligen Bootstrap-Informationen aus. Nach Erhalt dieser Informationen können beide Instanzen die zuvor gespeicherte Heartbeat-Nachrichten authentifizieren. Die Wahl des aktiven SDMs erfolgt implizit anhand der Instanz-*ID*, beispielsweise kann festgelegt werden, dass stets diejenige Instanz mit der höchsten (oder niedrigsten) *ID* aktiv ist. Das Auswahlkriterium muss ebenfalls systemweit bekannt sein. Alle SDM-Instanzen, welche das Kriterium nicht erfüllen, agieren im Folgenden passiv, senden jedoch weiterhin in regelmäßigen Abständen ihre jeweiligen Heartbeat-Nachrichten.

Das in Abschnitt 6.3.3 vorgestellte Service Discovery Protokoll läuft unverändert ab, wobei SOME/IP-SD Clients sich jeweils an denjenigen SDM wenden, dessen Heartbeat-Nachricht die höchste *ID* aufweist. Da Clients in der ersten Phase des Protokolls (6.6) eine (D)TLS-Verbindung zum SDM aufbauen, ist sichergestellt, dass kein Angreifer mit Netzwerkkontrolle als SDM auftreten kann[3].
Für das Erreichen der Ausfallsicherheit sind die folgenden Beobachtungen von Bedeutung:

1. Da dem aktiven SDM aus den Nachrichten (6.10) die Bootstrap-Informationen aller passiven SDM-Instanzen bekannt sind, kann er diese als Teil der BS_*-Liste in der zweiten Phase (siehe Nachricht 6.8) des Protokolls an alle Teilnehmer übertragen.
2. Passive SDM-Instanzen nehmen an dem SD-Protokoll teil und sind folglich in der Lage die übertragene BS_*-Liste lokal zu speichern.

Fällt der aktive SDM aus, gekennzeichnet durch das Ausbleiben seiner regelmäßigen Heartbeat-Nachrichten für ein bestimmtes Zeitfenster, gibt es einen eindeutigen Nachfolger: diejenige SDM-Instanz, welche nun die größte bzw. kleinste *ID* besitzt. Ein Neustart des sicheren SOME/IP-SD Protokolls ist nicht notwendig, da der neue SDM die Rolle des vorherigen nahezu nahtlos übernehmen kann.
Clients stellen das Ausbleiben der Heartbeat-Nachrichten des alten SDM ebenfalls fest und wenden sich folglich automatisch an die neue Instanz (z. B. um der Gruppe nach einem ECU-Neustart wieder beizutreten). Da die Bootstrap-Informationen des neuen SDM allen Empfänger bereits bekannt sind[4] kann dieser authentifizierte *sdControl*-Events

[3] Da dieser kein gültiges Zertifikat zur Authentifizierung vorweisen kann.
[4] Da diese in BS_* enthalten sind.

anstelle des alten SDM übertragen.
Lediglich die Fähigkeit zur Verwaltung des Schlüsselbaums geht dabei verloren, da der neue SDM nicht alle Teilschlüssel des Baums kennt, insbesondere nicht die individuellen Schlüssel der anderen Teilnehmer. Da die Geheimhaltungseigenschaft für das SOME/IP-SD Protokoll jedoch nur einen geringes Security Level besitzt, ist dies für einen Fallback-Mechanismus akzeptabel. Alternativ kann der jeweils aktive SDM auch die Sammlung der individuellen Schlüssel an die passiven SDM-Instanzen über die entsprechenden TLS-gesicherten Unicast-Verbindungen übertragen.

Der Ansatz ist offensichtlich anfällig gegenüber einem Angreifer mit vollständiger ECU-Kontrolle über eine ECU auf der eine SDM-Instanz allokiert ist. Durch geeignete Wahl der *ID* kann ein solcher Angreifer trivial die Rolle des aktiven SDM einnehmen und so das gesamte System kompromittieren. Somit erhöht der Ansatz zwar die Ausfallsicherheit für das System erheblich, *verringert* jedoch die Sicherheit des Gesamtsystems. Dies ist der Fall, da sich die mögliche Angriffsfläche für einen erfolgreichen Angriff von einer auf alle ECUs mit einer SDM-Instanz vergrößert.
Ein möglicher Ansatz dieser Gefahr zu begegnen ist der Einsatz einer Form der Mehrheitsentscheidung. Dazu agieren alle SDM-Instanzen als aktive SDMs und Clients können anhand der empfangenen BS_*-Listen eine Mehrheitsentscheidung treffen. Die übertragenen BS_*-Listen aller ehrlich agierenden Instanzen sind identisch, wodurch eine eventuell unehrlich agierende Instanz entdeckt werden kann.
Ein solcher Ansatz erhöht jedoch den Overhead für den Systemstart erheblich, da jeder Client das in Abschnitt 6.3.3 beschriebene Protokoll dabei für jede SDM-Instanz ausführen muss.
Da die Anforderungen für einen Angreifer vollständige ECU-Kontrolle zu erreichen jedoch sehr hoch sind, ist in der Praxis ein solch aufwendiger Ansatz nicht unbedingt notwendig.

7 Fazit

Das SOME/IP Konzept erlaubt durch Einsatz des Service Discovery Protokolls das dynamische Auffinden und Erstellen von Kommunikationskanälen zwischen Softwarekomponenten und somit die Entwicklung hochdynamischer, modularisierter Software. Die in Kapitel 4 durchgeführte Sicherheitsanalyse des Protokolls zeigt jedoch deutlich, dass aufgrund des Verzichts auf Sicherheitsmaßnahmen ein Angreifer ein auf SOME/IP basierendes System nahezu beliebig manipulieren kann. Die freie Verfügbarkeit von Werkzeugen zur Analyse und Verarbeitung von IP-basierter Kommunikation erlaubt es dabei selbst Angreifern ohne spezielles Fachwissen einfache Manipulationen durchzuführen. Aufgrund des geplanten Einsatzes für sicherheitskritische Systeme (z. B. ADAS) sind die möglichen Auswirkungen eines bösartigen Angriffs auf das System kaum hoch genug einzuschätzen.
Um Angriffe zu verhindern oder zumindest die Auswirkungen auf das Gesamtsystem im Fall eines erfolgreichen Angriffs auf ein Teilsystem zu reduzieren, wurden in Kapitel 5 eine Reihe von Sicherheitsmechanismen vorgestellt und auf deren Tauglichkeit hin untersucht.
Zuletzt wurde in Kapitel 6 ein Konzept zur Absicherung eines auf SOME/IP-basierten Systems mithilfe einer Kombination von (D)TLS und des TESLA-Verfahrens vorgestellt. Durch die strikte Absicherung aller SOME/IP Kommunikationskonzepte inklusive des SOME/IP-SD Protokolls kann eine sichere Kommunikation zwischen physikalisch getrennten Kommunikationspartnern erreicht werden. Durch den Einsatz von TESLA zur Sicherstellung der Senderauthentifizierung von Multicast-Nachrichten gilt dies insbesondere auch für das *worst-case* Szenario, in welchem ein Angreifer vollständige Kontrolle über eine ECU besitzt. Zwar kann die Kommunikation der vom Angreifer kontrollierten ECU nicht eingeschränkt werden, aber das vorgestellte Konzept verhindert eine Manipulation von Nachrichten, welche von nicht manipulierten ECUs gesendet wird.
Die Verwaltung eines Gruppenschlüssels kann für ein dynamisches System (wie es *adaptive* AUTOSAR darstellt) effizient mithilfe von Schlüsselbäumen erreicht werden. Bei Anwendung der entsprechenden Beitritts- bzw. Austrittsprozeduren (siehe Kapitel 5.4.2.3) kann sowohl *backwards secrecy* als auch *forward secrecy* für die so geschützte Kommunikation sichergestellt werden. Für diese Aufgabe und eine Gruppenzugangskontrolle ist ein zentraler, vertrauenswürdiger Kommunikationsknoten erforderlich. Ein solcher Knoten kann zusätzlich einen gesicherten Logging-Service ausführen, welcher alle im Netzwerk übertragenen Nachrichten und deren Empfangszeit aufzeichnet. Dies ermöglicht die Erfüllung der Verbindlichkeitseigenschaft für TESLA-geschützte Nachrichten (vgl. Kapitel 5.5.3.8). Mit Hinblick auf die Entwicklung autonomer Fahrzeuge

ist diese Eigenschaft von hohem Interesse, um im Falle eines Unfalls eine mögliche Manipulation des Systems nachzuweisen, analog zu der sogenannten Blackbox, welche für ähnliche Zwecke in der Luftfahrt eingesetzt wird.
Um einen *single point of failure* für das System zu vermeiden, kann zusätzlich ein redundantes System aus mehreren SDM-Instanzen verwendet werden, wodurch der Ausfall des SDM ausgeglichen werden kann.

Die Sicherheitsgarantien der vorgestellten Konzepte basieren dabei grundsätzlich auf der Verwendung von individuellen, herstellersignierten Zertifikaten für jede ECU, sowie der Trusted Plattform Annahme. Um die Sicherheit des Gesamtsystems gewährleisten zu können, müssen folglich beide Annahmen erfüllt sein. Eine Untersuchung dieser Aspekte liegt jedoch außerhalb des Fokus dieser Arbeit und stellt somit einen offensichtlichen Ansatzpunkt für weitere Untersuchungen dar. Sowohl der Aufbau, als auch die sichere Verwaltung einer herstellerspezifischen PKI, inklusive (teil-)automatischer Prozesse zur Ausstellung und Austausch temporärer Zertifikate, z. B. für freie Werkstätten oder Prüfstellen, stellt eine anspruchsvolle Aufgabe für die einzelnen Hersteller, Zulieferer und Werkstätten dar.
Gleiches gilt für die Implementierung einer Trusted Plattform, welche die in Kapitel 5.1 benannten Anforderungen erfüllt und als Grundlage für die *adaptive AUTOSAR* Plattform dienen kann.

Literaturverzeichnis

[1] S. Checkoway, D. McCoy, B. Kantor, D. Anderson, H. Shacham, S. Savage, K. Koscher, A. Czeskis, F. Roesner, T. Kohno, „Comprehensive experimental analyses of automotive attack surfaces“, in *Proceedings of USENIX Security Symposium*, 2011.

[2] E. Christmann, „Architektur, Aufgaben und Vorteile serieller Bussysteme“, *Datenkommunikation im Automobil*, Nr. 1, 2016.

[3] M. Broy, I. H. Krüger, A. Pretschner, C. Salzmann, „Engineering automotive software“, 2007.

[4] E. Christmann, „Sicherer Datenaustausch mit CAN“, *Datenkommunikation im Automobil*, Nr. 2, 2016.

[5] SAE, „Serial control and communication heavy duty vehilce network“, SAE International, Techn. Ber. J1939, 2013.

[6] R. N. Charlette. (2009). This car runs on code, IEEE Spectrum, Adresse: http://spectrum.ieee.org/transportation/systems/this-car-runs-on-code.

[7] K. Motors, „Automotive perspectives on cybersecurity“, presented at VDI Wissensforum 2017, Verein Deutscher Ingeneure (VDI), 2017.

[8] *Autosar*, Online, AUTOSAR Partnership, 2003. Adresse: http://www.autosar.org/.

[9] *AUTOSAR E-Learning*, E-Learning (Online), Vector Informatik GmbH, 2017. Adresse: https://elearning.vector.com/index.php?wbt_ls_kapitel_id=862081&root=376493&seite=vl_autosar_introduction_de.

[10] D. Becker, „Global automotive executive survey 2017“, KPMG, 18, 2017.

[11] *The mission and objectives of the CAR 2 CAR communication consortium*, Online.

[12] *V2v safety technology now standard on cadillac cts sedans*, Online, 9. März 2017. Adresse: http://media.cadillac.com/media/us/en/cadillac/news.detail.html/content/Pages/news/us/en/2017/mar/0309-v2v.html.

[13] (2016). Telematik-Tarife in der Kfz-Versicherung: Spion fährt mit, DPA, Adresse: www.berlin.de/special/auto-und-motor/kfz-versicherungen/4320318-961862-telematiktarife-in-der-kfzversicherung-s.html.

[14] *What is OnStar?, No matter where you're headed—onstar services keep you safe, connected and ready for the road ahead.* Online, General Motors, 2016. Adresse: https://www.onstar.com/us/en/home.html.

[15] *Bmw connected drive*, Online, BMW. Adresse: http://www.bmw.de/de/topics/faszination-bmw/connecteddrive/ubersicht.html.

[16] T. M. House, *Vehicle to grid (v2g), Wir integrieren Elektroautos in das Stromnetz*, Online. Adresse: http://www.mobilityhouse.com/de/vehicle-to-grid-und-vehicle-to-home/.

[17] „The adaptive platform for future use cases“, presented at Vector Congress 2016, 2016. Adresse: https://vector.com/congress/files/presentations/VeCo16_24_30Nov_Reithalle_Markl_Vector.pdf.

[18] „The AUTOSAR adaptive platform for connected and autonomous vehicles“, presented at 8th Vector Congress, S. Fürst, Hrsg., 2016.

[19] *AUTOSAR Adaptive Platform*, Online, Vector Informatik. Adresse: https://vector.com/vi_autosar_adaptive_de.html.

[20] *Open hardware for small spaces, Versatile single board computer*, Homepage, MinnowBoard Foundation, 2017. Adresse: https://minnowboard.org/.

[21] D. M. Oertel, „Adaptive AUTOSAR, Ready for Next Generation ECUs“, presented at Vector Technologie Tage, Vector Informatik, 18. Okt. 2017.

[22] M. Massoud, „Evaluation of an Adaptive AUTOSAR System in Context of Functional Safety Environments“, Master Thesis, Technische Universität Chemnitz, 14. Sep. 2017.

[23] K. Koscher, A. Czeskis, F. Roesner, S. Patel, T. Kohno, S. Checkoway, D. McCoy, B. Kantor, D. Anderson, H. Shacham, „Experimental security analysis of a modern automobile“, in *Proceedings of IEEE Symposium on Security and Privacy (SP)*, IEEE, 2010, S. 447–462.

[24] C. Miller, C. Valasek, „Remote exploitation of an unaltered passanger vehicle“, 2015.

[25] R. M. Ishtiaq Roufa, H. Mustafaa, S. O. Travis Taylora, W. Xua, M. Gruteserb, W. Trappeb, I. Seskarb, „Security and privacy vulnerabilities of in-car wireless networks: A tire pressure monitoring system case study“, in *19th USENIX Security Symposium, Washington DC*, 2010, S. 11–13.

[26] P. Kleberger, T. Olovsson, E. Jonsson, „Security aspects of the in-vehicle network in the connected car“, in *Proceedings of IEEE Intelligent Vehicles Symposium (IV)*, IEEE, 2011, S. 528–533.

[27] J. Weschke, F. Heeslund, „Testing and evaluation to improve data security of automotive embedded systems“, Magisterarb., Chalmers University of Technology, 2011.

[28] T. Hoppe, S. Klitz, J. Dittmann, „Security threats to automotive CAN networks - practical examples and selected short-term countermeasures“, in *In Proceedings of 27th International Conference on Computer Safety, Reliability and Security*, Bd. 8, 2008, S. 235–248.

[29] K. Strandberg, „Avoiding vulnerabilities in connected cars, A methodology for finding vulnerabilities“, Magisterarb., Chalmers University of Technology, 2016.

[30] D. Rieck, „Vehicle-2-X Kommunikation“, presented at 12. Carrier Meeting, Fraunhofer Institut für offene Kommunikationssysteme, 2013.

[31] C. Miller, C. Valasek. (2015). Hackers remotely kill a jeep on the highway — with me in it. A. Greenberg, Hrsg., Adresse: www.wired.com/2015/07/hackers-remotely-kill-jeep-highway/.

[32] A. Davis, „Broadcast your attack: security testing DAB radio in cars“, NCC Group, 2015. Adresse: https://www.nccgroup.trust/globalassets/resources/uk/presentations/2015/august/ncc-group-15-davis-broadcasting-your-attack-security-testing-dab-radio-in-cars.pdf.

[33] J. Buchmann, *Einführung in die Kryptographie*. Springer-Verlag, 2010.

[34] K. Schmeh, *Kryptographie, Verfahren, Protokolle, Infrastrukturen*, 6. Aufl. dpunkt Verlag, 2016, ISBN: 978-3-86490-356-4.

[35] „Kryptographische Verfahren: Empfehlungen und Schlüssellängen“, Bundesamt für Sicherheit in der Informationstechnik, Techn. Ber. BSI TR-02102-1, Version 2017-01, 2017.

[36] E. Barker, „Recommendation for key management“, National Institute of Standards and Technology (NIST), Techn. Ber. 800-57, Version 4, 2016, Kap. Part 1: General.

[37] D. E. Holcomb, W. P. Burleson, K. Fu, „Power-up SRAM state as an identifying fingerprint and source of true random numbers“, *IEEE Transactions on Computers*, Bd. 58, Nr. 9, S. 1198–1210, 2009.

[38] G. E. Suh, S. Devadas, „Physical unclonable functions for device authentication and secret key generation“, in *In Proceedings of 44th Annual Design Automation Conference*, ACM, 2007, S. 9–14.

[39] S. Zhao, Q. Zhang, Y. Quin, H. Guangyao, D. Feng, „Providing root of trust for ARM TrustZone using SRAM PUFs“, *International Association for Cryptologic Research*, 2014.

[40] E. Barker, J. Kelsey, „Recommendation for random number generation using deterministic random bit generators“, National Institute of Standards and Technology (NIST), Techn. Ber. NIST 800-90A, 2012.

[41] S. Bakhtiari, R. Safavi-Naini, J. Pieprzyk et al., „Cryptographic hash functions: A survey“, *Centre for Computer Security Research, Department of Computer Science, University of Wollongong, Australie*, 1995.

[42] H. Krawczyk, M. Bellare, R. Canetti, „Hmac: Keyed-hashing for message authentication“, Request for Comments, Techn. Ber. 2104, 1997.

[43] M. Dworkin, „Recommendation for block cipher modes of operation: the cmac mode for authentication“, NIST, Techn. Ber. NIST 300-38B, 2005.

[44] ISO/IEC, „Information technology - open system interconnection, The directory“, Public-Key and Attribute Certificate, International Organization for Standardization, Published Standard 26262, 2017, Kap. 8, S. 242. Adresse: https://www.iso.org/standard/43464.html.

[45] S. Farrell, R. Housley, S. Turner, „An internet attribute certificate profile for authorization“, Internet Engineering Task Force (IETF), Standards Track RFC 5755, Version 01, 2010.

[46] L. Rizzo, „Effective erasure code for reliable computer communication protocols“, *ACM Computer Communication Review*, 27 1997.

[47] *Die ADAC Pannenstatistik 2007*, Online, ADAC, 2007. Adresse: https://www.adac.de/_mmm/pdf/Pannenstat%202007_159KB_33406.pdf.

[48] *ADAC Pannenstatistik 2016*, Online, ADAC, 2016. Adresse: https://www.adac.de/%5C_mmm/pdf/28679_292063.pdf.

[49] *Zahl der Batterieausfälle steigt rasant, Von wegen wartungsfrei*, T-Online. Adresse: http://www.t-online.de/auto/technik/id_65917350/adac-statistik-autobatterien-fallen-immer-oefter-aus.html.

[50] „Example for a serialization protocol (SOME/IP)“, AUTOSAR, Standard 637, 2014.

[51] D. L. Löker, „SOME/IP service discovery“, presented at Vector Symposium, 2014.

[52] „Specification of service discovery“, AUTOSAR, Standard 616, 2016.

[53] ISO, „Road vehicles – functional safety“, International Organization for Standardization, Techn. Ber. 26262, 2011. Adresse: https://www.iso.org/standard/43464.html.

[54] *Verkehrssicherungspflicht - definition*, Online, Beratungsgesellschaft für Arbeits- und Gesundheitsschutz (BfGA). Adresse: https://www.bfga.de/arbeitsschutz-lexikon-von-a-bis-z/fachbegriffe-v-z/verkehrssicherungspflicht-fachbegriff.

[55] J. Sauler, S. Kriso. (2011). ISO 26262 - Die zukünftige Norm zur funktionalen Sicherheit von Straßenfahrzeugen. M. Hafner, Hrsg., Adresse: http://www.elektronikpraxis.vogel.de/themen/elektronikmanagement/projektqualitaetsmanagement/articles/242243/.

[56] I. 2. S. Group. (2017). ISO-SAE 21434 road vehicles – cybersecurity engineering, General overview, Adresse: %7Bhttps://wiki.unece.org/download/attachments/44269802/TFCS-05-12%20%28OICA-CLEPA%29%20Overview%20on%20ISO-SAE%20activities%20regarding%20Cyber%20Security.pdf?api=v2%7D.

[57] C. Schmittner, Z. Ma, C. Reyes, o. Dillinger, P. Puschner, „Using SAE J3061 for automotive security requirement engineering“, in *SAFECOMP 2016 Workshops*, 2016, S. 157–170.

[58] P. Folkesson, *Heavens*, Online. Adresse: http://www.sp.se/en/index/research/dependable_systems/heavens/Sidor/default.aspx.

[59] *The stride threat model*, Online, Microsoft, 2005. Adresse: https://msdn.microsoft.com/en-us/library/ee823878(v=cs.20).aspx.

[60] *Evita, E-safety vehicle intrusion protected applications*, Online, Frauenhofer SIT, 2008. Adresse: https://www.evita-project.org/.

[61] O. Henniger, L. Apvrille, A. Fuchs, Y. Roudier, A. Ruddle, B. Weyl, „Security requirements for automotive on-board networks“, in *Intelligent Transport Systems Telecommunications*, Bd. 9, 2009, S. 641–646.

[62] C. Ponikwar, H.-J. Hof, S. Gopinath, L. Wischhof, „Beyond the Dolev-Yao model: Realistic application-specific attacker models for applications using vehicular communication“, presented at International Conference on Emerging Security Information, Systems and Technologies - SECURWARE, Bd. 10, 2016.

[63] B. Kraft. (2017). Hacker brechen aus virtueller Maschine aus, Adresse: https://www.heise.de/newsticker/meldung/Hacker-brechen-aus-virtueller-Maschine-aus-3658416.html.

[64] W. Arbaugh, D. J. Farber, J. M. Smith et al., „A secure and reliable bootstrap architecture“, in *In Proceedings of 1997 IEEE Symposium on Security and Privacy*, IEEE, 1997, S. 65–71.

[65] „ARM security technology building a secure system using TrustZone® technology“, ARM Limited, Techn. Ber., 2005.

[66] R. Soja, „Automotive security: From standards to implementation“, NXP Semiconductors, White Paper, Version 1, 2014. Adresse: https://www.nxp.com/docs/en/white-paper/AUTOSECURITYWP.pdf.

[67] S. Mutumba, „Linux für Safety-kritische Anwendungen gesucht, Opensynergy unterstützt forschung von osadl“, 17. Aug. 2017.

[68] M. Ziehensack, „Safe and secure communication with automotive ethernet“, presented at Automotive Ethernet Technology Day 2015, Japan, 2015. Adresse: https://standards.ieee.org/events/automotive/2015/14_Safe_and_Secure_Communication_with_Automotive_Ethernet.pdf.

[69] „AUTOSAR Security Modules, Current status“, Vector Informatik, State-of-the-Art Review, 2015. Adresse: https://vector.com/portal/medien/solutions_for/Security/AUTOSAR_Security_Modules_Lecture_ESCAR_2015.pdf.

[70] E. Rescorla, T. Dierks, „The transport layer security (TLS) protocol version 1.2“, Internet Engineering Task Force (IETF), Standards Track RFC 5246, 2008.

[71] S. Kent, K. Seo, „Security architecture for the internet protocol“, Internet Engineering Task Force (IETF), Standards Track RFC 6347, 2005.

[72] „Media access control (MAC) security“, IEEE, Techn. Ber. IEEE 802.11AE, 2013.

[73] N. Ferguson, B. Schneier, „A cryptographic evaluation of ipsec“, *Counterpane Internet Security, Inc*, Bd. 3031, S. 14, 2000.

[74] E. Rescorla, „The transport layer security (TLS) protocol version 1.3“, draft-ietf-tls-tls13-21, Internet Engineering Task Force (IETF), Draft RFC 5246, Version 21, 2017.

[75] E. Rescorla, N. Modadugu, H. Tschofenig, „The datagram transport layer security (DTLS) protocol version 1.3, Draft-ietf-tls-dtls13-01“, Internet Engineering Task Force (IETF), Draft RFC 6347, Version 01, 2017.

[76] J. Salowey, H. Zhou, P. Eronen, H. Tschofenig, „Transport layer security (TLS) session resumption without server-side state“, Internet Engineering Task Force, Standard 5077, 2008. Adresse: https://tools.ietf.org/html/rfc5077.

[77] B. Dowling, M. Fischlin, F. Günther, S. Douglas, „A cryptographic analysis of the TLS 1.3 handshake protocol candidates“, 2017.

[78] R. Canetti, J. Garay, G. Itkis, D. Micciancio, M. Naor, B. Pinkas, „Multicast security: a taxonomy and some efficient constructions“, in *Proceedings of the 8th IEEE International Conference on Computer Communications (INFOCOM)*, IEEE, Bd. 2, 1999, S. 708–716.

[79] E. Bresson, O. Chevassut, D. Pointcheval, „Provably authenticated group diffie-hellman key exchange - the dynamic case“, 2001.

[80] A. T. Sherman, D. A. McGrew, „Key establishment in large dynamic groups using one-way function trees“, *IEEE transactions on Software Engineering*, Bd. 29, Nr. 5, S. 444–458, 2003.

[81] A. Perrig, R. Canetti, D. Song, J. D. Tygar, „Efficient and secure source authentication for multicast“, in *Proceedings of Network and Distributed System Security Symposium (NDSS)*, Bd. 1, 2001, S. 35–46.

[82] A. Pannetrat, R. Molva, „Efficient multicast packet authentication“, in *NDDS*, 2003.

[83] C. Karlof, N. Sastry, Y. Li, A. Perrig, J. Tygar, „Distillation codes and applications to DoS resistant multicast authentication“, in *NDSS*, Bd. 4, 2004, S. 37–56.

[84] A. Lysyanskaya, R. Tamassia, N. Triandopoulos, „Multicast authentication in fully adversarial networks“, in *Security and Privacy, 2004. Proceedings. 2004 IEEE Symposium on*, IEEE, 2004, S. 241–253.

[85] A. Perring, D. Song, R. Canetti, J. D. Tygar, B. B., „Timed efficient stream loss-tolerant authentication (TESLA): Multicast source authentication transform introduction“, Internet Engineering Task Force (IETF), Informational 4082, 2005.

[86] A. Perrig, R. Canetti, J. D. Tygar, D. Song, „The TESLA broadcast authentication protocol“, *RSA Cryptobytes*, Bd. 5, 2005.

[87] D. Liu, P. Ning, „Multilevel μTESLA: broadcast authentication for distributed sensor networks“, *ACM Transactions on Embedded Computing Systems (TECS)*, Bd. 3, Nr. 4, S. 800–836, 2004.

[88] *Using the gps system for accurate time*, Online, 10. Juli 2017.

[89] K. Behrendt, K. Fodero, „The perfect time: an examination of time-synchronization techniques“, 2005.

[90] N. O. Tippenhauer, C. Pöpper, K. B. Rasmussen, S. Capkun, „On the requirements for successful GPS spoofing attacks“, 2011.

[91] A. Perrig, „The BiBa one-time signature and broadcast authentication protocol“, in *Proceedings of the 8th ACM conference on Computer and Communications Security*, ACM, 2001, S. 28–37.

[92] *TLS Benchmarks, Hivemq 3.1.0 on aws*, Online, Benchmark Document, HiveMQ, 2016. Adresse: https://www.hivemq.com/blog/how-does-tls-affect-mqtt-performance/.

[93] *Cpuboss review*, Online, CPUBoss.com. Adresse: http://cpuboss.com/cpu/Intel-Atom-E3845.

Alle URLs wurden zuletzt am 14. November 2017 geprüft.

Erklärung

Ich versichere, diese Arbeit selbstständig verfasst zu haben. Ich habe keine anderen als die angegebenen Quellen benutzt und alle wörtlich oder sinngemäß aus anderen Werken übernommene Aussagen als solche gekennzeichnet. Weder diese Arbeit noch wesentliche Teile daraus waren bisher Gegenstand eines anderen Prüfungsverfahrens. Ich habe diese Arbeit bisher weder teilweise noch vollständig veröffentlicht. Das elektronische Exemplar stimmt mit allen eingereichten Exemplaren überein.

Ort, Datum, Unterschrift