Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Bachelorthesis

Leutrim Mustafa

Entwicklung eines Authentifizierungsstandards für Fahrzeug-zu-Fahrzeug / Fahrzeug-zu-Infrastruktur Kommunikationssysteme

Fakultät Technik und Informatik *Faculty of Engineering and Computer Science*

Leutrim Mustafa

Entwicklung eines Authentifizierungsstandards für Fahrzeug-zu-Fahrzeug / Fahrzeug-zu-Infrastruktur Kommunikationssysteme

Bachelorthesis eingereicht im Rahmen der Bachelorprüfung
im Studiengang Mechatronik
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer: Prof. Dr. Heike Neumann
Zweitgutachter: Dipl. Ing. Eckhard Walters

Abgegeben am 13. März 2017

Leutrim Mustafa

Thema der Bachelorthesis

Entwicklung eines Authentifizierungsstandards für Fahrzeug-zu-Fahrzeug/Fahrzeug-zu-Infrastruktur Kommunikationssysteme

Stichworte

Fahrzeug-zu-X Kommunikation (V2X), Authentifikation, Kommunikation, Zugangskontrolle, ITS-G5, IEEE 802.11p, Verschlüsselung, Signierung

Kurzzusammenfassung

Diese Arbeit beschäftigt sich mit der Entwicklung eines Authentifizierungsverfahrens für die Fahrzeug-zu-X-Kommunikation (V2X). Es wird eine Möglichkeit geschaffen, dass sich zwei Kommunikationspartner gegenseitig authentifizieren können, ohne ihre Identität preiszugeben. Für die Authentifizierung wird Challenge-Response-Verfahren mit digitaler Signatur verwendet. Dafür wird eine eigene Nachricht auf Basis der bestehenden Nachrichten definiert. Es wird ein Kommunikationsprotokoll konzipiert und implementiert. Zudem wird ein Demonstrator aufgebaut, welcher das entwickelte Authentifizierungsverfahren demonstriert.

Leutrim Mustafa

Title of the paper

Development of an authentication standard for vehicle-to-vehicle / vehicle-to-infrastructure communication systems

Keywords

Vehicle-to-X communication (V2X), authentication, communication, Access control, ITS-G5, IEEE 802.11p, encryption, signing

Abstract

This work deals with the development of an authentication method for vehicle-to-X communication (V2X). A possibility is created that two communication partners can mutually authenticate themselves without revealing their identity. The authentication uses challenge-response procedures with digital signature. A separate message is defined on the basis of the existing messages. A communication protocol is designed and implemented. In addition, a demonstrator is being developed, which demonstrates the developed authentication method.

Danksagung

Auf diesem Wege möchte ich mich bei all denjenigen bedanken, die mich während meines Studiums und besonders bei der Erstellung dieser Arbeit unterstützt haben.

Besonders bedanken möchte ich mich bei Eckhard Walters (NXP Semiconductors) für seine sehr kompetente Betreuung und seine Unterstützung beim Einarbeiten in die Thematik. Zudem möchte ich mich bei Peter Hierholzer (Cohda Wireless) und Jonas Vogt (htw Saar) bedanken, die mir während der gesamten Zeit für Fragen zur Verfügung standen und mich insbesondere in der Einarbeitungsphase mit sehr hilfreichen Tipps unterstützt haben.

Ein ganz besonderer Dank geht an meine Familie und meine Freunde, die mich während des gesamten Studium unterstützt und motiviert haben.

Inhaltsverzeichnis

Abkürzungsverzeichnis

Abkürzung	Bedeutung
ASN.1	Abstract Syntax Notation One
AU	Application Unit
BSM	Basic Safety Messages
BTP	Basic Transport Protocol
CAM	Cooparative Awareness Message
CCH	Controlchannel
C-ITS	Cooperative-Intelligent Transportation System
DENM	Decentralized Environmental Notification Messages
DOT	Departmend of Transportation
DSRC	Dedicated Short Range Communication
EPC	Electronical Product Code
ETSI	Europäische Institut für Telekommunikationsnormen
FSM	Finite State Machine
GNSS	Global Navigation Satellite System
iKoPA	integrierte Kommunikationsplattform für automatisierte Elektrofahrzeuge
IEEE	Institute of Electrical and Electronics Engineers
LLC Layer	Logical Link Control Layer
LTE	das Long Term Evolution
MAC-Layer	Media Access Control
MD5	die Message-Digest 5 mit 128 Bit
OBU	On-Board-Unit
Opcode	Operation Code
RFID	radio-frequency identification
RSU	Road-Side-Unit
SCH	Servicechannel
SDK	Software Development Kit
TCP	Transmission Control Protocol
UDP	User Datagramm Protocol
UMTS	Universal Mobile Telecommunications System
V2V	Vehicle2Vehicle oder kurz
V2X	Vehicle2Anything kurz
VM	Virtuellen Maschine
WAVE	Wireless Access for the Vehicular Environment
WLAN	Wireless Local Area Network
WSMP	WAVE Short Message Protocol

Abbildungsverzeichnis

Quellcodeverzeichnis

Tabellenverzeichnis

1 Einleitung

Automatisiertes Fahren ist eine neue Technologie, die den Transport von Menschen und Gütern verändern wird. Um vollautomatisiert fahren zu können, reicht es jedoch nicht aus, dass Fahrzeuge sich auf Ihre Sensordaten verlassen – sie müssen auch miteinander kommunizieren.

1.1 Motivation

Fahrzeuge können mit anderen Fahrzeugen und mit der Infrastruktur kommunizieren. So greifen Sensoren verschiedener Fahrzeuge und der Infrastruktur ineinander. Neben der Möglichkeit, Sensoren mit geringer Verzögerung miteinander zu fusionieren, kann die Kommunikation zwischen den Fahrzeugen aber auch für Interaktionen oder für Zugangskontrollen eingesetzt werden.

Heutzutage verfügen Fahrzeuge über sehr viele Sensoren, mit welchen sie ihre Umgebung erfassen. Diese Informationen werden meistens nur vom jeweiligen Fahrzeug ausgewertet. Die Reichweite von aktuellen Fahrzeugsensoren, wie LIDAR, Radar, Ultraschall und Kameras ist eingeschränkt, da sie auf direkte Sicherverbindungen bauen.
Um sicher vollautomatisch fahren zu können, muss das Fahrzeug seine Umgebung so weit wie möglich kennen.

Durch die Kommunikation von Fahrzeugen untereinander und mit ihrer Umgebung können Sensordaten anderen Fahrzeugen zur Verfügung gestellt werden. So kann ein Fahrzeug frühzeitig eine Gefahr erkennen, bevor diese überhaupt in Sichtweite des Fahrers oder Fahrzeuges ist. Auch können Fahrzeuge selbständig andere Verkehrsteilnehmer über Unfälle, Staus und weitere Gefahrenstellen warnen.

Durch die Vernetzung der Fahrzeuge untereinander und mit ihrer Umgebung werden die Verkehrssicherheit und die Verkehrseffizienz erhöht. Voraussetzung hierbei ist jedoch, dass die Daten auch verlässlich sind und nur für den vorgesehenen Zweck verwendet werden. Datensicherheit und Datenschutz sind essentiell.

1.2 Ziel dieser Arbeit

Ziel dieser Arbeit ist es, eine Möglichkeit zu entwickeln, dass Fahrzeuge sich gegenüber anderen Fahrzeugen oder der Infrastruktur authentifizieren[1] können. Hierdurch soll erreicht werden, dass Fahrer und Fahrzeug eine Berechtigung für eine Aktion nachweisen ohne ihre Identität preiszugeben. Dies wird mittels Verschlüsselung und Anwendung eines Pseudonyms oder einer Gruppenidentität erreicht.

Im vorliegenden Anwendungsfall wurde als Aktion ein Zugang zu einer Dienstleistung (elektrisches Laden, Parken) gewählt. Für die Datenübertragung soll die Fahrzeug-zu-Fahrzeug Kommunikation genutzt werden. Nach vorheriger Registrierung und Anmeldung bei einem Dienstleister, soll sich ein Fahrzeug selbständig authentifizieren, wenn es sich in der Nähe der Zugangskontrolle beim Zielort befindet. Dies bedeutet, dass das Fahrzeug den Zugangspunkt automatisch erkennt und eine Authentifizierung startet.

Aktuell ist dies für die Fahrzeug-zu-Fahrzeug Kommunikation nicht standardisiert. Daher ist die Arbeit so aufgebaut, dass sie durch die Industrie in Standardisierungsgremien eingebracht werden kann. Die Standardisierung selber ist nicht Teil der Bachelorarbeit.

[1] Beweis der Identität erbringen [27].

2 Grundlagen

Die drahtlose Kommunikation ist in heutigen Fahrzeugen schon Stand der Technik. So sind Bluetooth, das Universal Mobile Telecommunications System (UMTS), das Long Term Evolution (LTE) und das Global Navigation Satellite System (GNSS) fast schon serienmäßig in den Fahrzeugen integriert. Cellulare Systeme wie UMTS und LTE werden für die Nutzung von nicht sicherheitskritischen Anwendungen genutzt, z.B. für Komfort- und Infotainmentanwendungen.

Für sicherheitskritische Anwendungen sind die Anforderungen weit komplexer, da Daten aus der drahtlosen Kommunikation Einfluss auf die Sicherheit des Fahrzeugs nehmen. Daher werden in diesem Kapitel die Grundlagen der V2X-Kommunikation, die Standards und Sicherheitsaspekte der Kommunikation erläutert. Insbesondere muss für kooperative Sicherheitsanwendungen auf garantierte maximale Latenzzeiten geachtet werden.

2.1 V2X Kommunikation

Die Kommunikation zwischen zwei Fahrzeugen wird "Vehicle to Vehicle-Communication" (Vehicle2Vehicle oder kurz V2V) genannt. Die Kommunikation zwischen einem Fahrzeug und der Infrastruktur wird "Vehicle to Infrastructure-Communication" (Vehicle2Infrasturcture oder kurz. V2I) genannt. Allgemein wird die Kommunikation von Fahrzeugen "Vehicle to Anything-Communication" (Vehicle2Anything kurz V2X) genannt [1].

Mit V2X ist ein Informations- und Datenaustausch zwischen Fahrzeugen und ihrer Umgebung gemeint. Das Ziel einer V2X-Kommunikation ist es andere Verkehrsteilnehmer frühzeitig über Gefahren und Risiken zu warnen.

Heutzutage sind die Fahrzeuge mit zahlreichen Sicherheitssystemen und Sensoren ausgestattet, mit Hilfe dieser erfassen sie die Umgebung und Informationen über den aktuellen Fahrzustand. Diese Daten werden vom Fahrzeug zusammengefasst und über V2X verbreitet. Somit können Fahrzeuge andere Verkehrsteilnehmer über mögliche Gefahrenstellen, wie Glatteis auf der Straße, Aquaplaning, Staus, Unfälle oder andere gefährliche Verkehrssituationen warnen, bevor diese mit ihnen konfrontiert werden.

Die Informationsübermittlung erfolgt über verschiedene Funksysteme nach "Institute of Electrical and Electronics Engineers (IEEE)-Standard", so z.B. das IEEE 802.11p oder die Mobilfunkstandards LTE und UMTS für den Datenaustausch über das Mobilfunknetz. Bei 802.11p handelt es sich um einen Wireless Local Area Network (WLAN) Standard, welcher speziell für Fahrzeuge konzipiert wurde [2]. Neben der garantierten minimierten Latenzzeit ist 802.11p auch dadurch charakterisiert, dass es auch ohne Infrastruktur funktioniert, also direkt von Fahrzeug zu Fahrzeug funktioniert.

Damit ein Fahrzeug über V2X kommunizieren kann, benötigt es eine sogenannte On-Board-Unit (OBU). Die OBU sammelt Informationen über die eigene aktuelle Geschwindigkeit, Position, Richtung, Lenkeinschlag, etc. und sendet diese über V2X an andere Verkehrsteilnehmer. Sie ist somit die Schnittstelle zwischen Fahrer, Fahrzeug und Umgebung.

Die V2X Kommunikation wird auch für die Verkehrslenkung genutzt. Die sogenannten Road-Side-Units (RSUs) sind an Lichtsignalanlagen, Warnschildern oder Autobahnbrücken stationär montiert und können über V2X mit den Fahrzeugen kommunizieren. Sie bilden eine Kommunikationsinfrastruktur entlang der Verkehrswege und sind untereinander oder auch mit dem Internet verbunden. Die RSUs kommunizieren über 802.11p direkt mit den OBUs der Fahrzeuge und liefern den Fahrzeugen aktuelle Verkehrsinformationen und dienen somit der Verkehrsüberwachung und -Steuerung [2].

Die Fahrzeuge können, wenn sie sich in entsprechender Nähe befinden, über 802.11p miteinander kommunizieren. Die Reichweite einer 802.11p Verbindung liegt je nach Topologie der Umgebung zwischen 500 und 1000 Meter. Dabei werden die Informationen von einem Fahrzeug gesendet und alle in dem Empfangsbereich des Sendefahrzeugs empfangen diese Informationen.

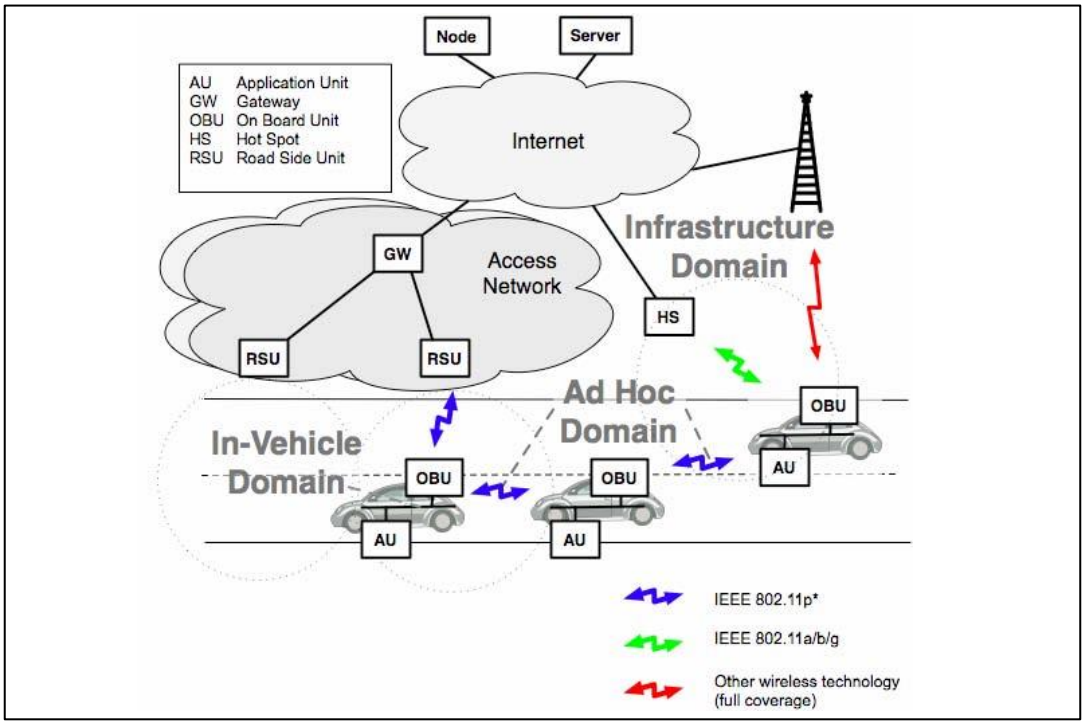

Abbildung 2.1 - V2X Architektur [2]

In Abbildung 2.1 ist die Architektur von V2X dargestellt.
Man erkennt, dass OBUs untereinander und zu den RSUs über ein Ad-hoc-Netzwerk[2] kommunizieren. Die OBU bekommt Daten aus einer Steuereinheit des eigenen Fahrzeugs, einer sogenannten AU (Application Unit) und verbreitet diese über 802.11p. Jedes V2X-fähige Gerät im Funkbereich kann diese Daten empfangen und verwerten. Die RSUs sind über Gateways[3] mit dem Internet verbunden und können darüber auch eine Verbindung zwischen dem Fahrzeug und dem Internet herstellen.

[2] Funknetz, welches aus zwei oder mehreren Geräten besteht und sich selbst aufbaut [26].
[3] Ein Gateway verbindet Rechnernetze, welche auf Unterschiedlichen Protokollen basieren [43].

2.2 ASN.1

Unter ASN.1 (Abstract Syntax Notation One) versteht man eine abstrahierte Notation für Datenstrukturen. Mit Hilfe von ASN.1 ist es möglich komplexe Datenstrukturen, Datentypen und Elemente von Nachrichten und Protokollen abstrakt, sprachenunabhängig und plattformunabhängig zu beschreiben.

Bei ASN.1 wird die Struktur einer Nachricht abstrakt definiert. Durch einen ASN.1 Compiler[4] wird die Nachricht in eine Programmiersprache umgewandelt. Die so erzeugten Dateien können im Quellcode mit eingebunden und in ein ausführbares Programm umgewandelt werden.

Abbildung 2.2 zeigt den Ablauf von der Erstellung eines Protokolls in ASN.1 bis zum ausführbaren Programm [3].

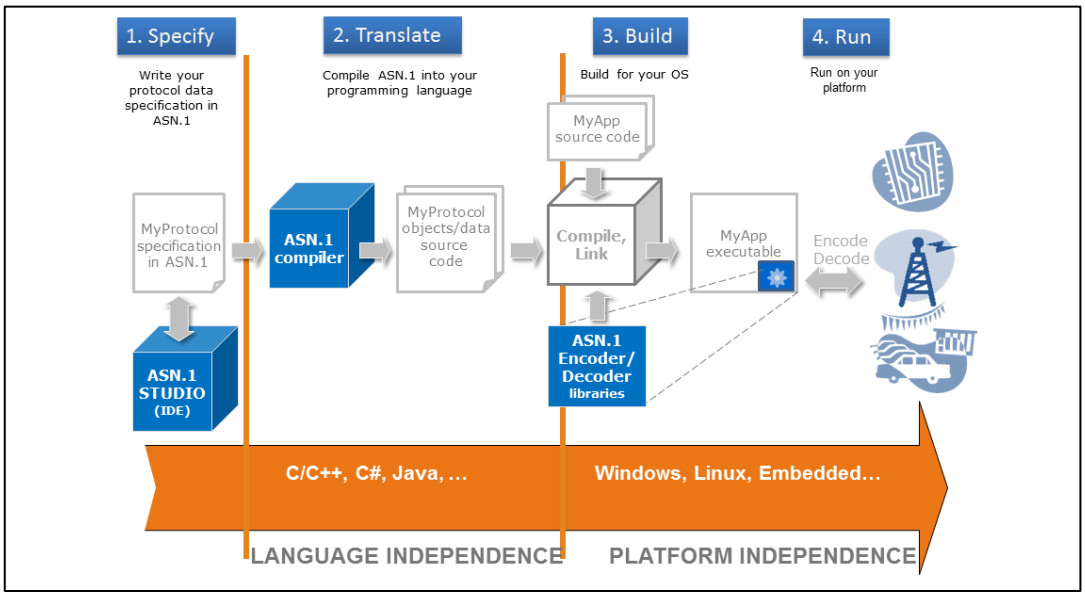

Abbildung 2.2 - ASN.1 Compiler Ablauf [3]

In ASN.1 gibt es verschiedene Datentypen. Die wichtigsten sind in Tabelle 2.1 zusammengestellt. Es sind weitaus mehr Datentypen vorhanden, welche je nach Einsatzzweck genutzt werden können.

Tabelle 2.1 - ASN1 Datentypen [4]

TYP	BESCHREIBUNG
BOOLEAN	Variable mit zwei Zuständen, TRUE oder FALSE
INTEGER	Integer Variable, Darstellung ganzzahliger Werte
BIT STRING	Binäre Daten jeglicher Art
OCTET STRING	Binäre Daten die durch 8 teilbar sind
PRINTABLESTRING	Groß- und Kleinbuchstaben und Sonderzeichen als String

[4] Computerprogramm zum Übersetzten von Quellcode in eine Programmsprache [29].

```
1       -- Machine Control Abstract Syntax
2        MCAS DEFINITIONS::=
3       BEGIN
4       ControlPacket ::= SEQUENCE {
5              currentStatus INTEGER {off(0), on(1)},
6              nextAction    INTEGER {noAction(0), turnOn(1), turnOff(2)}
7       }
8       END
```

Code 2.1 - ASN1 Bespiel [3]

In Code 2.1 - ASN1 Bespiel ist ein einfaches Kontrollprotokoll zur Maschinensteuerung in ASN.1 definiert.

Die Nachricht beinhaltet die Struktur „ControlPacket“. In dieser sind „currentStatus“ und „nextAction“ definiert. „currentStatus“ ist als ein Integerwert[5] definiert und kann den Wert „0“ für Off oder „1“ für On annehmen.
„nextAction“ ist auch ein Integerwert und kann die Werte „0“ für noAction, „1“ für turnOn oder „2“ für turnOff annehmen.

[5] Datentyp zum Speichern ganzzahliger Werte [33].

2.3 ISO/OSI-Modell

Das OSI-Modell (Open System Interconnection) wurde von der ISO (International Organization for Standardization) als Referenzmodell für Kommunikationssysteme entwickelt [5].
Das Modell ist hierarchisch aufgebaut und besteht aus sieben Schichten. Diese sind im Folgenden aufgelistet und in Abbildung 2.3 grafisch dargestellt.

7) Anwendungsschicht (Application Layer)
Funktionen für Anwendungen

6) Darstellungsschicht (Presentation Layer)
Ausgabe von Daten im geeigneten Format

5) Sitzungsschicht (Session Layer)
Steuerungs- und Kontrollmechanismen für den Datenaustausch

4) Transportschicht (Transport Layer)
Logische Ende-zu-Ende Verbindung

3) Vermittlungsschicht (Network Layer)
Datenflusskontrolle

2) Sicherungsschicht (Data Link Layer)
Fehlererkennungsmechanismen

1) Bitübertragungsschicht (Physical Layer)
Elektrische, mechanische und funktionale Schnittstelle zum Übertragungsmedium

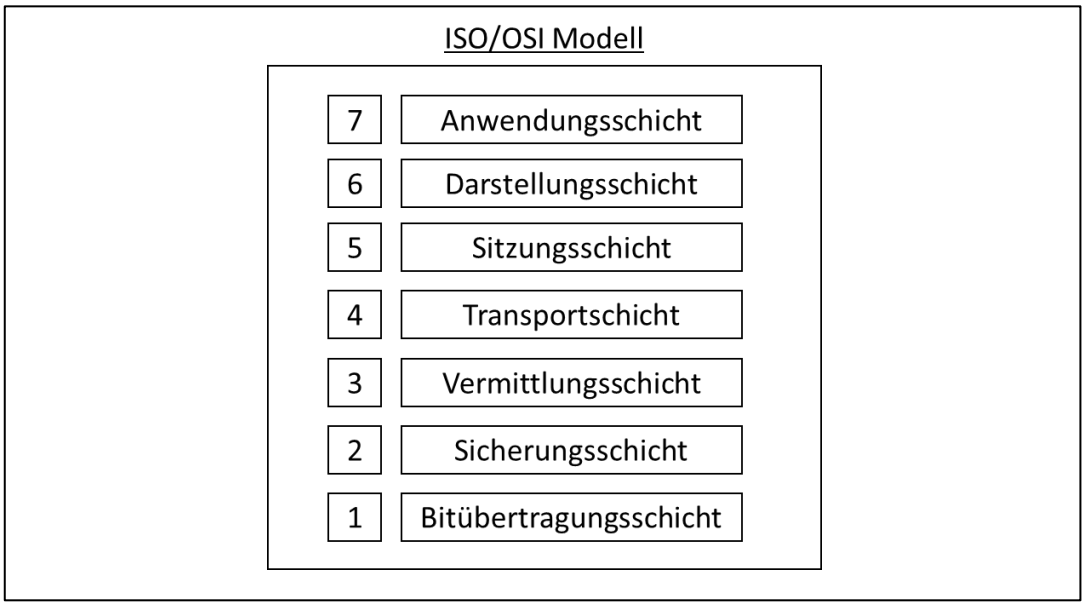

Abbildung 2.3 - ISO/OSI Modell [5]

2.4 IEEE 802.11p

IEEE ist die Abkürzung für “Institute of Electrical and Electronics Engineers” und ist ein weltweiter Berufsverband von Ingenieuren. Die IEEE bildet Gremien für die Standardisierung von Software, Hardware und Techniken. Eines dieser Standards ist die 802.11 Norm für die Kommunikation in Funknetzwerken.

802.11 wird auch als „Wireless Local Area Network“ (WLAN) bezeichnet. Es spezifiziert im OSI-Modell den Mediumszugriff (MAC-Layer) und die physikalische Schicht (Physical-Layer). Zur Datenübertragung werden Funkspektren im Frequenzband bei 2,4 GHz und 5 GHz genutzt.

802.11a, b, g, n, ac sind Varianten von 802.11 und wurde für den stationären Einsatz entwickelt. Sie sind nicht für hochdynamische Umgebungen gedacht. 802.11p ist eine Erweiterung von 802.11a. Die -p Variante wurde für den Einsatz in Fahrzeug-zu-Fahrzeug-Netzwerken, also für Mobilität optimiert [6]. Mit 802.11p soll die Verkehrssicherheit erhöht werden, sodass spezielle Anforderungen, wie die hohe Zuverlässigkeit und niedrige Latenz erfüllt werden. Beim 802.11p Standard wurde im MAC-Layer eine Priorisierung der Nachrichten hinzugefügt. So können Notfallnachrichten priorisiert abgearbeitet werden. Daneben ist 802.11p besonders geeignet, schnell Verbindungen aufzubauen und weiterzuleiten.

Tabelle 2.2 - Vergleich 802.11a und 802.11p [7]

	IEEE 802.11A	**IEEE 802.11P**
FREQUENZBAND	5,180 – 5,825 GHz	5,850 – 5,925 GHz
BANDBREITE	20 MHz	5, 10, 20 MHz
DATENRATE	6, 9, 12, 18, 24, 36, 48, 54 Mbps	3, 4.5, 6, 9, 12, 18, 24, 27 Mbps
MODULATION	BPSK OFDM QPSK OFDM 16-QAM OFDM 64-QAM OFDM	BPSK OFDM QPSK OFDM 16-QAM OFDM 64-QAM OFDM
SYMBOLDAUER	4,0 µs	8,0 µs
SICHERHEITSABSTAND	0,8 µs	1,6 µs

In Tabelle 2.2 erkennt man, dass 802.11p das Frequenzband oberhalb von 802.11a nutzt [7]. Dies liegt zwischen 5,850 und 5,925 GHz. Bei der Bandbreite wurde 802.11p um 5 und 10 MHz erweitert. Die Datenrate hingegen wurde auf 27 Mbps begrenzt. Zudem wurden die Symboldauer und der Sicherheitsabstand verdoppelt. Dies führt zu einer höheren Robustheit und zuverlässigeren Übertragung.
Aufbauend auf 802.11p wurde vom US-amerikanischen Departmend of Transportation (DOT) die Dedicated Short Range Communication (DSRC) entwickelt. Daraus haben sich verschieden Varianten für die USA, Japan und Europa entwickelt.

- USA: WAVE
- Europa: ETSI ITS G5
- Japan: Japan-Forum

Im Folgenden wird näher auf die amerikanischen und europäischen Standards eingegangen, da diese für die Entwicklung wichtig sind. Der japanische Standard würde über den Rahmen dieser Arbeit hinausgehen und wird daher nicht näher betrachtet.

2.5 WAVE

In den USA wurde vom IEEE der US spezifische Protokollstack[6] WAVE (Wireless Access for the Vehicular Environment) standardisiert.
In Abbildung 2.4 wird der Aufbau des WAVE Protokollstack aufgezeigt.

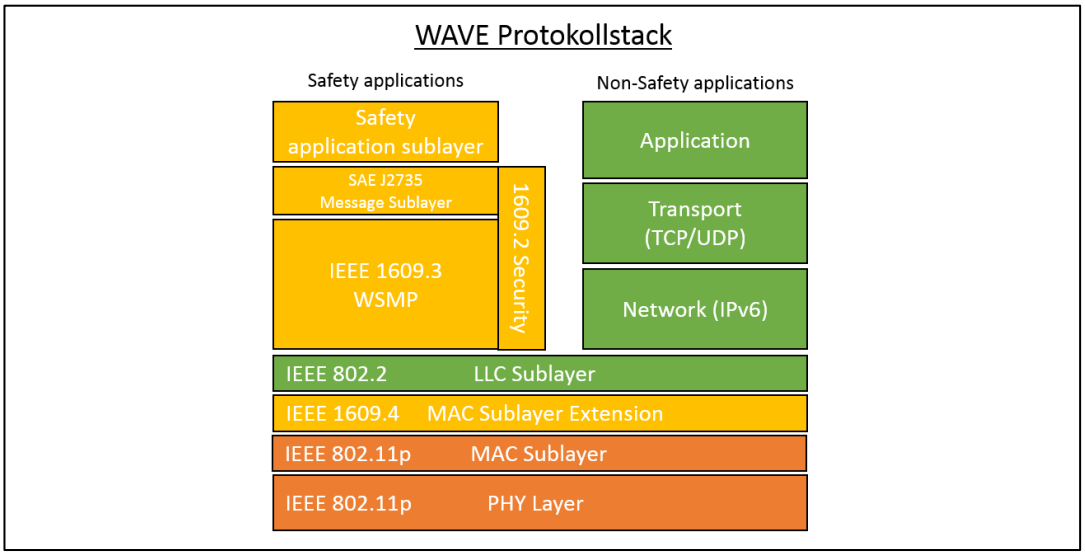

Abbildung 2.4 - WAVE Protokollstack [45]

Im PHY Layer (Physicale-Layer) und MAC-Layer wird der 802.11p Standard genutzt. Darauf aufbauend werden die darüber liegenden Schichten durch die IEEE 1609 Protokolle definiert. Ab dem Logical Link Control Layer (LLC Layer) unterteilen sich die Schichten in einen „Safety“ Zweig und „Non-Safety“ Zweig.

Der Non-Safety Zweig zeigt die Schichten für nicht sicherheitsrelevanten Anwendungen und Nachrichten der Komfort- und Infotainmentanwendungen.

Der Safety-Zweig repräsentiert die sicherheitsrelevante Kommunikation, mit den Basic Safety Messages (BSM), welche in dem SAE J2735 Message Directory definiert sind.

Im IEEE 1609.4 – Multi-Channel Operation ist die Aufteilung der Kommunikation auf verschiedene Kanäle definiert [8]. Im 5,9 GHz Bereich sind 7 Kanäle zu je 10 MHz vorgesehen. Sechs dieser Kanäle sind Servicekanäle, auch Service channels (SCH) genannt. Einer davon ist ein Steuerkanal, dieser wird auch Control channel (CCH) genannt. Die Aufteilung dieser Kanäle ist in Abbildung 2.5 gezeigt.

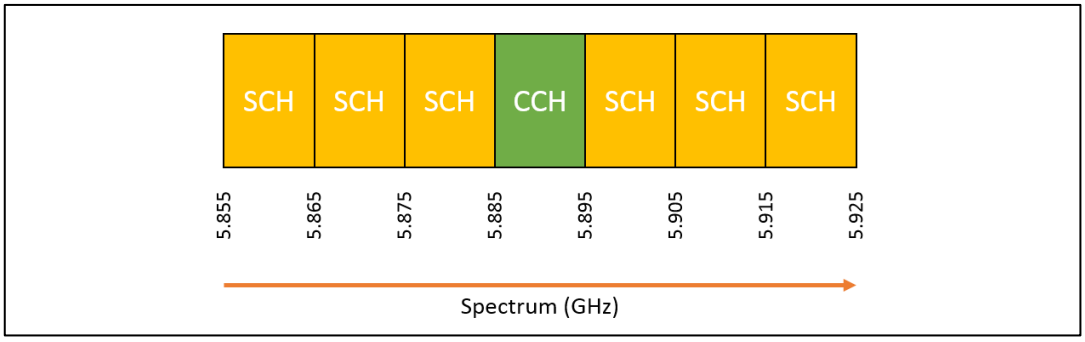

Abbildung 2.5 - WAVE Frequenzbandnutzung [37]

[6] Stapel an Protokollen die miteinander im Bezug stehen [38].

Im IEEE 1609.3 - Networking Services ist das WSMP (WAVE Short Message Protocol) definiert [9]. Im WSMP ist vorgesehen, dass die sicherheitsrelevanten Nachrichten nur mit einem sehr geringen Overhead versehen werden.

Im IEEE 1609.2 sind alle sicherheitsrelevanten Maßnahmen für die Kommunikation definiert [10].

2.6 ETSI C-ITS G5

Das Europäische Institut für Telekommunikationsnormen (ETSI) ist eine Normungsorganisation für Informations- und Kommunikationstechnologien in Europa.

Das ETSI standardisiert die V2X-Kommunikation aufbauend auf dem 802.11p Standard in dem Cooperative-Intelligent Transportation System (C-ITS) [11].
In Abbildung 2.6 ist aus dem ETSI Dokument [11] der C-ITS Protokollstack dargestellt. Man erkennt, dass sechs verschiedene Blöcke dargestellt sind, wovon drei sich in das ISO/OSI Modell einteilen lassen.

Im untersten Block ist die Accessschicht, in der die Bitübertragung stattfindet. Darüber findet sich der Networking & Transport Block. In diesem werden die Daten in ein Protokoll zusammengefügt und für das Verschicken vorbereitet.

Die Facilities-Schicht sitzt darüber und stellt die Daten für die Protokolle zur Verfügung. Ganz oben befindet sich der Applications-Block, in welcher die eigentliche Anwendung definiert ist.

Diese aufgeschichteten Blöcke werden von der Security und Management Schicht eingerahmt. Dort wird die Verschlüsselung der Datenpakete und die Auslastung der verschiedenen Übertragungskanäle verwaltet.

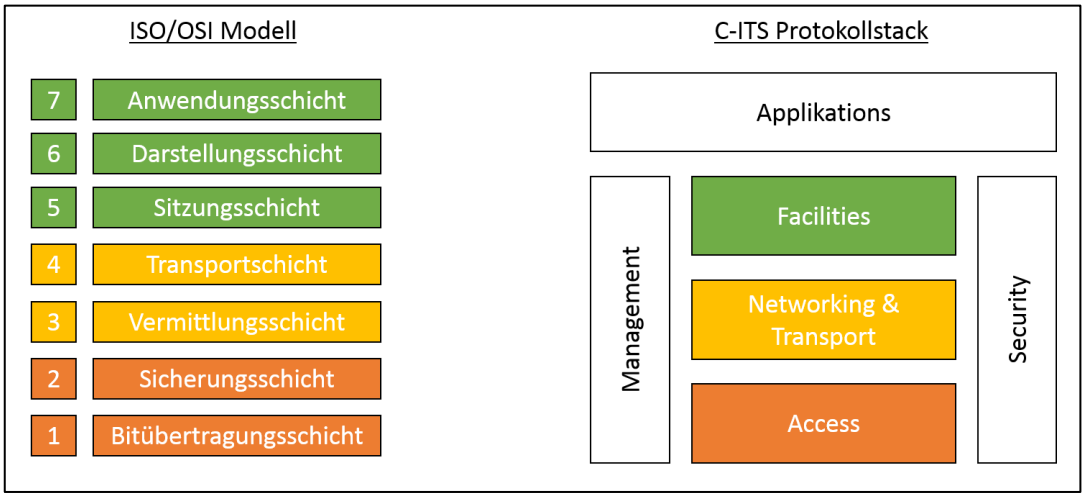

Abbildung 2.6 - C-ITS Protokollstack [11]

2.6.1 Access

In dieser Schicht sind nach dem ISO/OSI Modell die Bitübertragungs- und Sicherungsschicht definiert. Es werden verschiedene drahtlose Kommunikationstechnologien unterstützt. Darunter auch der 802.11p Standard, welcher jedoch für den europäischen Raum angepasst wurde und hier ITS-G5 genannt wird.

2.6.2 Networking & Transport

In dieser Schicht sind die Vermittlungs- und Transportschicht nach dem ISO/OSI Modell definiert. In der Transportschicht sind Protokolle, wie das Transmission Control Protocol (TCP) und das User Datagramm Protocol (UDP) vorgesehen. Zudem wurden Erweiterungen, wie das Basic Transport Protocol (BTP), eingearbeitet. Das BTP ermöglicht Ende-zu-Ende Verbindungen innerhalb eines V2X-Netzwerks.

Die Vermittlungsschicht sieht Netzwerkprotokolle wie IPv4 und IPv6 vor. Diese Schicht wurde um das Geonetwork erweitert. Beim Geonetworking Protokoll ist eine geografische Adressierung von Nachrichten möglich [11]. Auch hier wird wieder der Mobilität der Anwendungen Rechnung getragen.

Geonetworking
Durch das Geonetworking ist eine Möglichkeit geboten, Nachrichten geografisch zu adressieren und weiterzuleiten. Dadurch können Nachrichten auch auf einen geografischen Bereich beschränkt werden. Es ist somit keine direkte Adressierung eines Teilnehmers notwendig. Alle Teilnehmer, welche sich in dem Geltungsbereich der Nachricht befinden, empfangen diese.

2.6.3 Facilities

Diese Schicht ist mit der Sitzungsschicht des ISO/OSI Modells gleichzusetzen. Hier werden Nachrichten wie die Cooparative Awareness Message (CAM) und die Decentralized Environmental Notification Messages (DENM) definiert. Die Informationen für diese Nachrichten werden in dieser Ebene generiert und zusammengestellt. Die Nachrichten werden mit Zeitstempeln und Positionsdaten versehen. In dieser Ebene wird auch geregelt, ob die Nachrichten periodisch oder ereignisbasiert generiert werden.

CAM
Eine CAM wird periodisch von jedem sich im ITS-Netzwerk befindlichen Gerät verschickt. In einer CAM werden unterschiedliche Informationen, wie zum Beispiel die Zeit, die Position und der Bewegungszustand des Fahrzeuges gesendet. Dazu beinhaltet die CAM weitere Informationen, wie den Fahrzeugtyp, die Fahrzeugdimensionen und die Rolle im Straßenverkehr (Auto, Krankenwagen, Polizeiwagen, etc.). Dadurch bekommen die ITS-Teilnehmer ein genaues Bild ihrer Umgebung und der sich in der Nähe befindlichen Teilnehmer.
Der Aufbau einer CAM wird im Abschnitt 2.6.7 näher beschrieben, da die CAM eine große Rolle in dieser Arbeit spielt [12].

DENM
Eine DENM wird durch ein bestimmtes Ereignis ausgelöst und dient dazu andere Verkehrsteilnehmer zu benachrichtigen. Einige dieser Ereignisse wären zum Beispiel ein Stau, Glatteis auf der Straße, ein Unfall und ähnliche Ereignisse, welche das Verkehrsgeschehen beeinträchtigen [13].

2.6.4 Applications

Die Application-Ebene ist in die drei ITS-Anwendungen, „Road Safety“, „Traffic Efficiency“ und „Other Applications“, aufgeteilt.

Die Road Safety – Anwendung ist für die Vermeidung von Unfällen zuständig. Sie warnt den Fahrer frühzeitig vor (un-)vermeidbaren Unfällen, Gefahrenstellen und greift bei autonomen Fahrzeugen in die Fahrzeugsteuerung ein [11].

Die Traffic Efficiency – Anwendung dient der Optimierung des Verkehrsflusses, wie der Vermeidung von Staus. Sie kommuniziert unter anderem mit Lichtsignalanlagen und sorgt für einen effizienten Fluss des Verkehrs.

Unter „Other Applications“ gehören Anwendungen, wie zum Beispiel Zugangskontrollen oder Zahlungsdienste.

2.6.5 Management

In der Management-Ebene wird die Verwaltung der verschiedenen Anwendungen geregelt. In dieser Ebene wird auch die Auslastung der Kanäle gesteuert. So wird hier die Frequenz des Versands von Datenpakete gesteuert [11].

2.6.6 Security

In der Security-Schicht werden sicherheitsrelevante Dienste, welche für die Kommunikation wichtig sind, bereitgestellt. Die Verwaltung der Firewall und die Erkennung von unerlaubten Zugriffen, sowie deren Vermeidung, gehören zu den Aufgaben der Security-Schicht [11].

2.6.7 CAM Aufbau

```
1       CAM ::= SEQUENCE {
2             header  ItsPduHeader,
3             cam     CoopAwareness
4       }
5
6       CoopAwareness ::= SEQUENCE {
7         generationDeltaTime GenerationDeltaTime,
8         camParameters CamParameters
9       }
10
11      CamParameters ::= SEQUENCE {
12       basicContainer BasicContainer,
13       highFrequencyContainer HighFrequencyContainer,
14       lowFrequencyContainer LowFrequencyContainer OPTIONAL,
15       specialVehicleContainer SpecialVehicleContainer OPTIONAL,
16       ...
17      }
```

Code 2.2 - CAM Definition ASN1 [12]

Die Struktur einer CAM-Nachricht besteht aus zwei Blöcken, dem „header“ mit dem Namen „ITSPduHeader“ und der „cam“ mit dem Namen „CoopAwareness“ **[12]**. Die „CoopAwareness“ ist eine Sequenz aus der generationDeltaTime und dem „camParamters“. „camParamteres“ ist wiederrum eine Sequenz und beinhaltet den „basicContainer“, den highFrequencyContainer“, den lowFrequencyContainer“ und den „speicalVehicleContainer“ [Code 2.2].

In diesen „Containern“ sind die weiteren Parameter definiert und werden von der Software mit Informationen gefüllt.

```
1       ItsPduHeader ::= SEQUENCE {
2             protocolVersion INTEGER{currentVersion(1)} (0..255),
3             messageID INTEGER{denm(1),cam(2), poi(3), spat(4), map(5), ivi(6),
4                               ev-rsr(7)} (0..255),
5             stationID StationID
6       }
```

Code 2.3 - ItsPduHeader [12]

In dem „ITSPduHeader“ sind die Informationen über die „protocolVersion“, die „messageID“ und „stationID“ [Code 2.3] definiert.

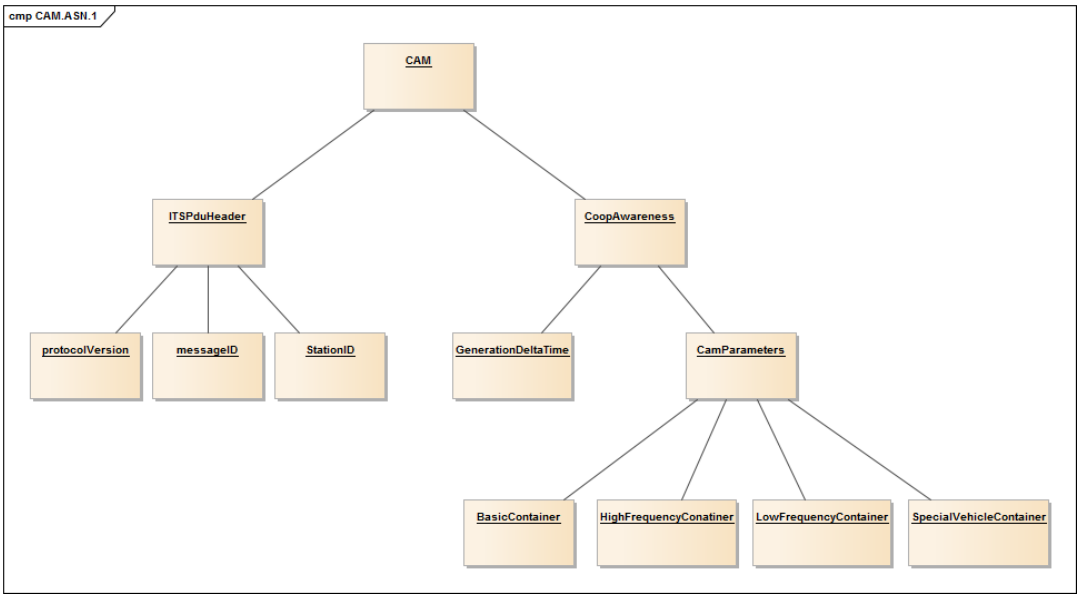

Abbildung 2.7 - CAM Aufbau [24]

2.7 Sicherheit

Im deutschen Sprachgebrauch kann Sicherheit sowohl die Bedeutung des englischen Begriffs *„Security“* als auch den des Begriffs *„Safety“* beinhalten. Nachfolgend wird auf die Bedeutung entsprechend der *„Security“* betrachtet, also den Schutz vor Angriff durch nicht kooperative Benutzer. „Safety“ hingegen adressiert Aspekte der Robustheit und der Vermeidung ungewollter Störungen und wird hier nicht näher betrachtet.
Sicherheit im Sinne einer Vermeidung unerlaubter Eingriffe spielt bei der V2X Kommunikation eine große Rolle, da durch unbefugten Zugriff, ein direkter Eingriff in den Straßenverkehr und das Fahrverhalten eines Fahrzeuges erfolgen kann. Daher muss die V2X-Kommunikation vor unbefugten Zugriff geschützt werden und das Manipulieren von Daten verhindert werden.

2.7.1 Authentifikation

Damit eine Person oder ein Benutzer bestimmte Handlungen ausführen kann, die nur er durchführen darf, muss er beweisen, dass er hierzu berechtigt ist.

In digitalen System wird eine Zugangsberechtigung in drei Schritte unterteilt [14].

- Zunächst wird die Identität angegeben, dies nennt man Identifikation.
- Danach wird der Beweis dieser Identität erbracht, die sogenannte Authentifikation.
- Den Beweis der damit verbunden Rechte, nennt man Autorisation.

In dieser Arbeit geht es insbesondere um die Authentifikation und Autorisation, also den Beweis einer Identität und der damit verbunden Rechte. Aus Datenschutzgründen soll die Identität nicht preisgegeben werden, da ansonsten ein Rückverfolgen des Benutzers möglich wäre. Daher wird die Identität durch ein Pseudonym ersetzt, welches über bestimmte Recht verfügt.

Den Nachweis der eigenen Identität kann man nach [14] in drei verschiedene Arten klassifizieren.

- Wissen (Information besitzen wie z.B. Pin oder Passwort)
- Besitz (z.B. Schlüssel oder Chipkarte)
- Eigenschaft (biometrische Merkmale wie z.B. Fingerabdruck)

Jede Nachweisart hat seine Vor- und Nachteile und ist jeweils für verschieden Anwendungsszenarien geeignet.

Wissen
Die wissensbasierte Authentifikation ist die weit verbreitete Methode. Dabei werden Passwörter in textueller Form genutzt und meistens an eine Benutzerkennung geknüpft. So ist gewährleistet, dass nur die Person, die das Wissen über Benutzerkennung und Passwort hat, Zugriff zu geschützten Bereichen erhält. Zudem identifiziert sich der Nutzer mit seiner Benutzerkennung und somit ist eine eindeutige Zuordnung möglich. Ein großer Nachteil an wissensbasierten Authentifizierungen ist, dass das Wissen dupliziert und weitergegeben werden kann. So können auch Dritte Zugang erhalten und sich als autorisierter Nutzer ausgeben.

Besitz
Bei der besitzbasierten Authentifizierung hat der Nutzer meistens einen physikalischen Schlüssel oder eine digitale Signatur, mit dessen Besitz er beweisen kann, dass er ein autorisierter Nutzer ist. Digitale Signaturen sind vergleichbar mit Unterschriften. Den „Besitz“ gibt es nur einmal und dieser sollte nicht dupliziert werden können. Bei digitalen Zertifikaten oder Signaturen wird dies durch ein eindeutiges Schlüsselpaar erreicht. Diese bestehen aus einem öffentlichen und privaten Schlüssel. Mit dem privaten Schlüssel können Dateien signiert werden und mit dem öffentlichen Schlüssel kann diese Signatur überprüft werden.

Eigenschaft
Die eigenschaftsbasierte Authentifizierung ist meistens an biometrische Eigenschaften bzw. Merkmale einer Person gebunden. So ist das Duplizieren nur sehr schwer möglich und eine eindeutige Identifikation gegeben, da die Merkmale nicht an Dritte weitergegeben werden können. Merkmale, die typischerweise für Authentifizierungen genutzt werden, sind z.B. Iris-Muster, Fingerabdrücke oder Sprechtonhöhe. Theoretisch können alle eindeutigen biologischen Merkmale einer Person für eine Authentifikation genutzt werden.

2.7.2 Kryptografische Hash-Funktion

Der Hashwert[7] einer Datei ist gleichzusetzen mit einem Fingerabdruck. Die Hash-Funktion berechnet aus einem beliebig langen Datensatz eine Zeichenkette mit fester Länge. Für die kryptografischen Hash-Funktionen gibt es drei besondere Anforderungen [15].

- **Eindeutigkeit**
  Bei identischer Zeichenfolge muss die Hash-Funktion den selben Hashwert liefern.
- **Reversibilität**
  Die Ursprungsdatei darf nicht aus dem Hashwert berechnet werden können.
- **Kollisionsresistenz**
  Zwei unterschiedliche Datensätze dürfen nicht den gleichen Hashwert haben.

Es gibt viele verschieden Hash-Funktion. Eine weit verbreitete Funktion ist die Message-Digest 5 mit 128 Bit (MD5). Inzwischen gilt dieses Verfahren nicht mehr als sicher. Für den Anwendungszweck in dieser Arbeit reicht das MD5-Verfahren jedoch aus.

Aus dem Dokument „How to Break MD5 an other Hash Functions“ [16] geht hervor, dass die Schwachstelle von MD5 in der Kollision von Hashwerten liegt. Das bedeutet, dass es möglich, dass zwei unterschiedliche Datensätze den selben Hashwert ergeben.

Nach [17] benötigt man etwa 2^{39} Rechenoperationen um eine Kollision zu finden. Das entspricht in etwa 10 Stunden Rechenzeit. Die Authentifizierung über V2X läuft in wenigen Sekunden ab, sodass nicht genügend Zeit wäre eine Kollision zu finden und diese als Schwachstelle zu nutzen. Zudem wird bei jeder Authentifizierung eine neue Zufahlszahl generiert, sodass jedes Mal ein neuer Hashwert entsteht.

Ein Angriff ist theoretisch möglich, in der Praxis jedoch nicht umsetzbar. Zudem wird das Konzept so aufgebaut, dass die kryptographischen Funktionen einfach ausgetauscht werden können.

[7] Prüfsumme fester Länge einer Datei [31].

2.7.3 Digitale Signatur

Eine digitale Signatur für Dateien ist ähnlich wie eine persönliche Unterschrift auf einem Vertrag. Für eine digitale Signatur ist ein privater Schlüssel zum Signieren und ein öffentlicher Schlüssel zum Validieren nötig. Anhand von Abbildung 2.8 wird erklärt, wie das signieren einer Datei abläuft [18].

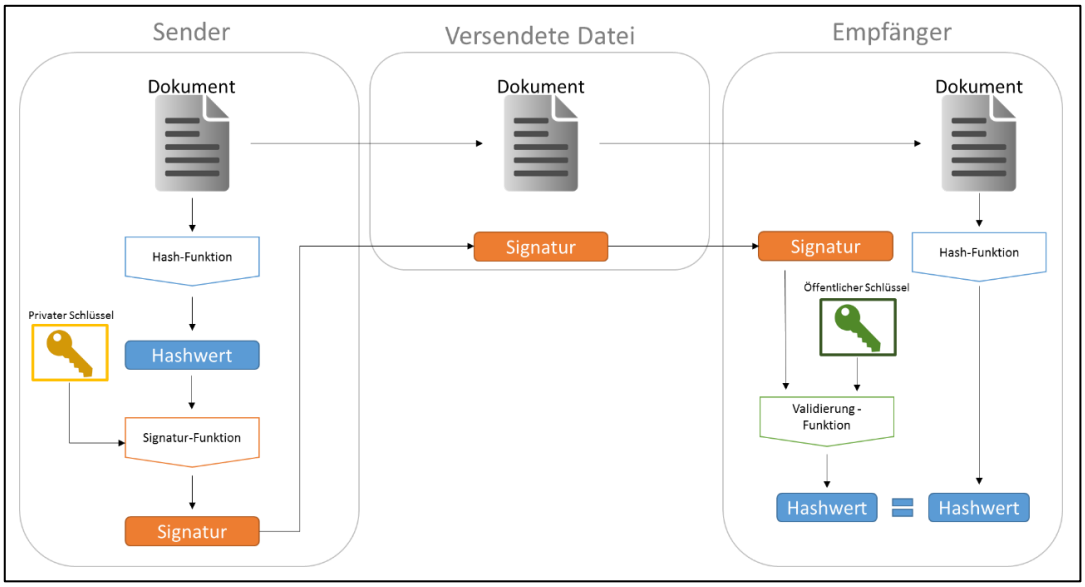

Abbildung 2.8 - Ablauf digitale Signatur [25]

Auf der einen Seite befindet sich der Sender, welcher ein Dokument versenden möchte. Auf der anderen Seite ist der Empfänger, der das Dokument empfängt und überprüfen möchte, ob das empfangende Dokument tatsächlich von seinem Kommunikationspartner, dem Sender, kommt. Hier muss sowohl die Fälschung des Dokumentes als auch das Abstreiten durch den Sender vermieden werden.

Es ist sehr aufwendig das komplette Dokument zu signieren. Daher wird mit Hilfe einer Hash-Funktion ein Hashwert berechnet. Dieser wird mit dem privaten Schlüssel des Senders signiert.
Diese Signatur wird dem Dokument hinzugefügt und beides wird gemeinsam an den Empfänger geschickt. Der Empfänger berechnet auch den Hashwert des Dokuments und überprüft die Signatur mit Hilfe des öffentlichen Schlüssels des Senders.

Dabei muss der berechnete Hashwert vom Empfänger mit dem Hashwert aus der Signatur identisch sein. Nur dann kann sich der Empfänger sicher sein, dass das Dokument tatsächlich von seinem Kommunikationspartner kommt und nicht auf dem Weg zu ihm verändert wurde. Der Empfänger darf auch nicht in der Lage sein, den Hashwert selber zu erzeugen, sonst ist die Signatur wertlos (Schutz gegen Abstreitbarkeit).

2.7.4 Challenge-Response-Verfahren

Im „Handbook of Applied Cryptography“ [19] ist beschrieben, wie das Challenge-Responseverfahren mit digitaler Signatur für eine Authentifizierung genutzt wird.

> *„Unilateral authentication with random numbers: Reliance on timestamps may be replaced by a random number, at the cost of an additional message:*
>
> $$A \leftarrow B : r_B \quad (1)$$
> $$A \rightarrow B : cert_A, r_A, B, S_A(r_A, r_B, B) \quad (2)$$
>
> *B verifies that the cleartext identifier is its own, and using a valid signature public key for A (e.g., from $cert_A$), verifies that A's signature is valid over the cleartext random number r_A, the same number r_B as sent in (1), and this identifier. The signed r_A explicitly prevents chosen-text attacks."*

(A) möchte sich bei (B) authentifizieren. (B) schickt eine Challenge an (A). Diese kann ein Zeitstempel oder eine Zufallszahl sein. Bei einem Zeitstempel ist es wichtig, dass (A) und (B) dieselbe Zeit nutzen, da ansonsten die Authentifizierung fehlschlägt.

In diesem Fall wird zur Laufzeit eine Zufallszahl (r_B) als Challenge generiert (2). Die Generierung der Zufallszahl zur Laufzeit ist wichtig, damit bei jedem Authentifizierungsverfahren eine andere Zufallszahl gesendet wird. Dadurch ist gewährleistet, dass die Challenge nicht vorhersehbar und damit reproduzierbar ist und manipuliert oder mehrfach abgespielt werden kann. Zudem kann (B) überprüfen, ob die Antwort zu der Challenge gehört, die er generiert hat.

(A) empfängt die Challenge und generiert seinerseits eine Zufallszahl (r_A) und fügt diese der erhaltenden Challenge hinzu. Zudem wird die Identität von (B) hinzugefügt. Das Hinzufügen von einer weiteren Zufallszahl und der Identität beugt sogenannte „Chosen-Text-Attacken“ vor, bei denen durch Senden verschiedener Challenges versucht wird, auf den privaten Schlüssel zu schließen.

B, r_A und r_B werden gemeinsam mit dem privaten Schlüssel von (A) signiert (S_A). Damit (B) diese Signatur überprüfen kann, muss der Signatur die Zufallszahl im Klartext und die Identität von (B) hinzugefügt werden. Sollte (B) den öffentlichen Schlüssel von (A) nicht besitzen, wird dieser als ($cert_A$) auch hinzugefügt.

Das Ganze wird als Response (2) zurück an (B) gesendet. (B) kann überprüfen, ob diese Response für ihn gedacht ist und dann die Signatur überprüfen.

3 Rahmenbedingungen

In diesem Kapitel werden die Rahmenbedingungen dargestellt. Es wird auf die MK5 OBU von Cohda Wireless eingegangen, welche für die Kommunikation über ITS G5 genutzt wird. Auch wird das Förderprojekt iKoPA erklärt, in dessen Rahmen ein Demonstrator aufgebaut werden soll. Die Authentifizierung über V2X soll, wenn möglich, ähnlich dem RFID Authentifizierungsverfahren sein. Daher wird in diesem Kapitel auch das RFID Konzept erklärt. Wo eine konzeptionelle Verbesserung gegenüber dem RFID Stand der Technik möglich ist, wird aber bewusst von dem RFID Verfahren abgewichen.

3.1 Cohda Wireless

Cohda Wireless ist ein Anbieter von Hard- und Software im Bereich der Intelligenten Verkehrssysteme. Das Unternehmen stellt Hardware- und Softwareprodukte für die Kommunikation über V2X her. Eins dieser Produkte ist die MK5 OBU, welche den RoadLink Chipsatz von NXP nutzt. Dieser Arbeit wird in Zusammenarbeit mit NXP verfasst, sodass die Auswahl der OBU auf eine, welche die NXP Technologie nutzt, fällt. Cohda Wireless bietet sowohl die MK5 Box als auch einen Protokollstack an.

3.1.1 MK5 - Hardwareaufbau

Die Cohda Wireless MK5 OBU bildet die Schnittstelle zwischen dem Fahrzeug und der V2X-Welt.
Für die ITS G5 Kommunikation verfügt die MK5 über zwei Anschlüsse für 802.11p Antennen. Für die Positionsbestimmung ist ein Anschluss für eine GNSS-Antenne vorhanden.

Abbildung 3.1 - Cohda MK5 Seitenansicht 1 [28]

In Abbildung 3.2 sieht man die Seite der Cohda MK5 mit den drei Antennenanschlüssen. Zudem erkennt man einen Mikro-SD-Karten Einschub. Dort kann eine SD-Karte eingelegt werden, auf welcher Log Dateien gespeichert werden. Diese können nach einer Session betrachtet und ausgewertet werden.

Abbildung 3.2 - Cohda MK5 Seitenansicht 2 [21]

In Abbildung 3.2 ist die andere Seite der Cohda MK5 dargestellt. Man erkennt dort einen Anschluss für die Spannungsversorgung. Die MK5 kann sowohl mit +12V als auch mit +24V betrieben werden.

Zudem ist ein Sub-D Anschluss zu sehen. Dieser kann für eine CAN-BUS[8] Kommunikation (Controller Area Network) zum Fahrzeug genutzt werden. Daneben befindet sich ein Ethernet-Anschluss. Dieser ist für die Kommunikation mit einem Notebook geeignet. Darüber kann die Software auf die MK5 geladen werden. Natürlich kann der Ethernet Connector auch zum Koppeln an die Fahrzeugelektronik oder zum Anbinden an das Internet verwendet werden, wenn die Applikationssoftware dies vorsieht.

Zudem besitzt die MK5 einen USB-Anschluss, welcher auch für eine Kommunikation mit einem Notebook genutzt werden kann.
Zum Anschluss eines externen Lautsprechers verfügt die MK5 über einen Klinkenausgang.

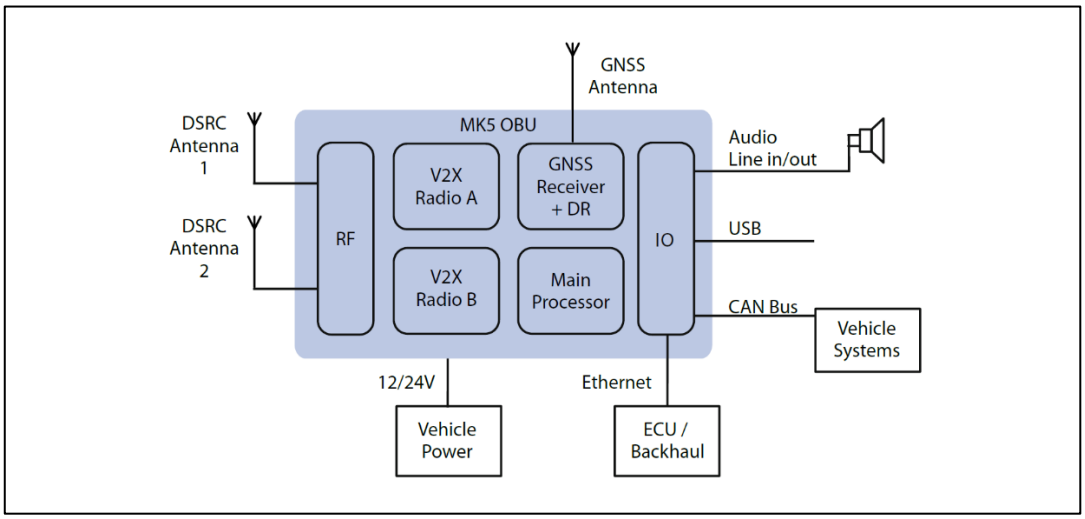

Abbildung 3.3 - Schematischer Aufbau MK5 [28]

[8] Serielles Bussystem

In Abbildung 3.3 ist ein schematischer Aufbau der Komponenten in der MK5 zusehen. Man erkennt die beiden DSRC-Antennen, welche an dem RF-Modul angeschlossen sind. Es wird der RoadLink[9] Chipsatz von NXP für V2X Kommunikation genutzt. Als RF-Modul wird der NXP TEF5100 (WISPA-ITS) und als Baseband-Modul, der NXP SAF5100 (MARS-ITS) genutzt [20].

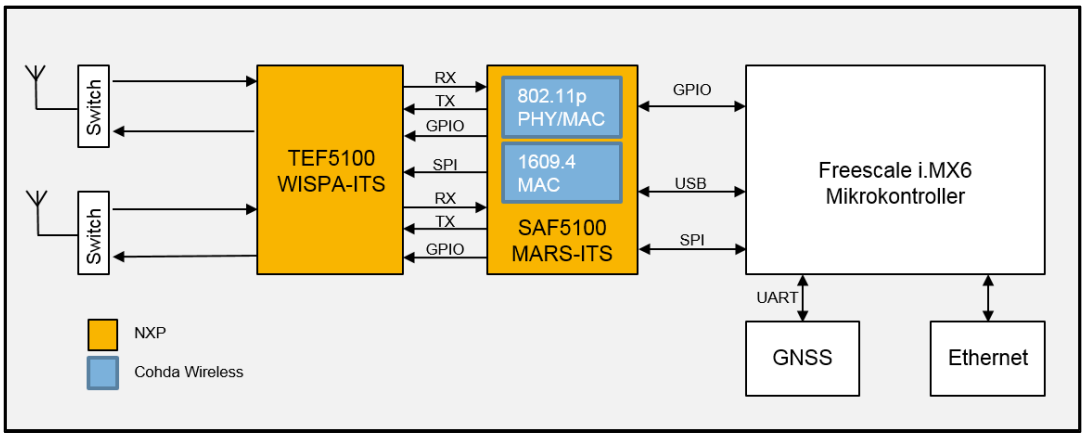

Abbildung 3.4 - RoadLink Chipsatz [39]

In Abbildung 3.4 ist die Kommunikation zwischen dem WISPA-ITS, dem MARS-ITS und dem Mikrokontroller dargestellt. Die Abbildung wurde basierend auf dem Datenblatt des SAF5100 vereinfacht erstellt. Das Datenblatt ist firmenintern und darf daher nicht veröffentlicht werden. Die beiden 802.11p Antennen sind an dem WISPA-ITS angeschlossen. Im WISPA-ITS werden die Signale vorverarbeitet und an den MARS-ITS weitergeleitet. Im MARS-ITS findet die Auswertung auf der MAC/LLC-Ebene statt.
Als Prozessor ist der Freescale i.MX6 verbaut, der nach der Übernahme des Freescale Unternehmens durch NXP nun auch zum NXP Portfolio gehört. Außerdem ist ein GNSS-Receiver verbaut, welcher für die Positionsbestimmung genutzt wird. Für die Kommunikation zum Fahrzeug sind Ethernet, CAN-Bus und USB vorhanden.

3.1.2 Cohda Wireless SDK

Von Cohda Wireless wird eine Entwicklungsumgebung, auch „Software Development Kit“ (SDK) genannt, in Form einer Virtuellen Maschine (VM) für Entwickler zur Verfügung gestellt. Auf die SDK können nur von Cohda freigeschaltete Entwickler zugreifen [21]. Daher wird in diesem Kapitel nur kurz auf die SDK eingegangen und die Funktionalität nicht näher erläutert.
Die SDK basiert auf dem Linux Ubuntu 14.04 Betriebssystem und beinhaltet alle benötigten Bibliotheken und Compiler für die Entwicklung von Software für die Cohda Produkte. Sie setzt auf den Cohda Protokoll-Stack auf, der bereits die Softwareebenen abdeckt.

Die VM wird in die Software Virtual Box eingebunden und kann als eigenständiges Betriebssystem genutzt werden.

In der SDK kann die Software für die Cohda MK5 kompiliert und über eine SCP-Verbindung auf die Cohda MK5 übertragen werden.

[9] Chipsatz für V2X Kommunikation von NXP Semiconductors [20].

3.2 iKoPA

„iKoPA“ steht für „Integrierte Kommunikationsplattform für automatisierte Elektrofahrzeuge“ und ist ein Förderprojekt des Bundesministeriums für Bildung und Forschung.
Das Förderprojekt wurde am 01.12.2015 gestartet und hat eine Laufzeit von drei Jahren.

3.2.1 Projektpartner

Zu den Projektpartnern gehören:

- Fraunhofer SIT, Darmstadt
- Fraunhofer FOKUS, Berlin
- DCAITI, Berlin
- MyOmega System Technologies, Nürnberg (nach Insolvenz ausgeschieden)
- SWARCO Traffic Systems, München
- NXP Semiconductors Germany, Hamburg
- ULD Schleswig-Holstein, Kiel
- BMT GmbH, München
- htw Saar, Saarbrücken

3.2.2 Ziele

Für hochautomatisierte Fahrfunktionen von Elektrofahrzeugen und einem effizienten elektrischen Fahren, ist eine transparente Kommunikationsstruktur notwendig. Wie in der Vorhabenbeschreibung aufgeführt, ist es Ziel von iKoPA eine einheitliche Kommunikation für die Elektromobilität zu schaffen. Es sollen verschiedene Infrastruktursysteme mit unterschiedlichen Kommunikationsstandards miteinander verbunden werden.

Für eine effiziente Elektromobilität ist ein Zusammenspiel von intelligenten Verkehrssystemen, intelligenten und automatisieren Elektrofahrzeugen sowie intelligenten Energie-Infrastrukturen unumgänglich.

3.2.3 iKoPA Szenario

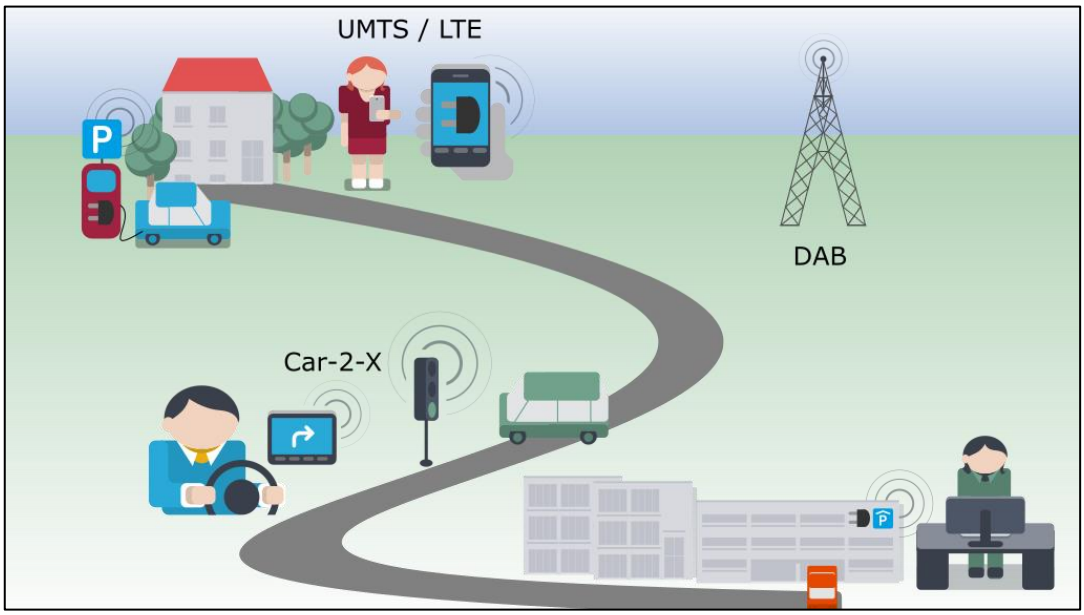

Abbildung 3.5 - iKoPA Szenario [32]

Ein Beispiel für die Anwendung der Ergebnisse aus iKoPA ist eine automatisierte Fahrt eines Elektrofahrzeugs zu einem Ladeparkplatz [Abbildung 3.5].
Dabei reserviert der Fahrer mit seinem Smartphone über Mobilfunk einen Parkplatz mit Ladestation in einem Parkhaus an seinem gewünschten Zielort. Danach setzt sich der Fahrer in sein Fahrzeug und fährt los.

Auf der Strecke zum Parkhaus kommuniziert das Fahrzeug mit der Infrastruktur über V2X oder hier auch Car-to-X genannt. Die Lichtsignalanlagen erkennen mittels V2X, Kameras und weiteren Sensoren den Verkehrsfluss und können somit individuell und direkt auf verschiedene Verkehrssituationen reagieren. Dem Fahrzeug wird eine verzögerungsfreie Fahrt zum Parkplatz ermöglicht. Mehrere hundert Meter vor dem Parkhaus beginnt eine erste Authentifizierung über V2X. Dadurch weiß das Parkhaus, dass das Fahrzeug demnächst an der Schranke des Parkhauses ankommen wird. Dies hilft für eine effiziente Einfahrtskontrolle und kann als Voranmeldung betrachtet werden.

Wenn sich das Fahrzeug nun unmittelbar vor der Schranke befindet, wird diese entweder bereits aufgrund der Authentifizierung per V2X und die Positionserkennung geöffnet, oder es erfolgt eine Identifizierung durch den RFID Leser. Der Leser verbirgt hierbei die Identität des Fahrzeuges. Dafür besitzt das Fahrzeug einen RFID-Transponder (engl.: RFID-TAG), welcher im Kennzeichen eingearbeitet wurde oder auf der Windschutzscheibe unlösbar verklebt ist. War die Authentifizierung erfolgreich, wird der Fahrer gebeten auszusteigen und das Fahrzeug fährt automatisch zum richtigen Parkplatz mit oder ohne Ladesäule. Die Konzeption und die Implementierung sehen somit verschieden Ausgestaltungsvarianten vor. So kann eine große Zahl verschiedener Ausführungsformen von Parkhäusern, Parkplätzen und Ladestellplätzen bedient werden.
Für die Orientierung und Routenplanung innerhalb des Parkhauses erhält das Fahrzeug über V2X die Kamerabilder des Parkhauses. Am richtigen Parkplatz angekommen, wird das Laden des Fahrzeuges vollautomatisch von der Ladesäule gestartet. Nach Beendigung des Lade- oder Parkvorganges wird das Fahrzeug wieder automatisch an der Schranke abgeliefert.

3.3 RFID Konzept

RFID steht für „radio-frequency identification“ und beschreibt eine Technologie, mit welcher eine berührungslose und automatische Identifizierung möglich ist.

3.3.1 Funktionsweise

Ein RFID-Transponder, auch RFID-Tag genannt, besteht prinzipiell aus einer Antenne, einem analogen Schaltkreis für das Senden und Empfangen, sowie einem digitalen Schaltkreis oder Mikrocontroller und permanentem Speicher.
Im Speicher sind Informationen über den Tag und dessen Identität gespeichert.

Das RFID-Lesegerät erzeugt ein hochfrequentes elektromagnetisches Wechselfeld, welchem der RFID-Tag ausgesetzt wird. Über die Antenne wird die hochfrequente Energie aufgenommen und dient während der Übertragung als Stromversorgung für den Mikrokontroller auf dem RFID-Tag.

Die vom Lesegerät gesendeten Befehle werden vom Mikrocontroller decodiert. Die Antworten werden in das eingestrahlte elektromagnetische Feld moduliert. Der RFID-Tag schwächt oder reflektiert das Feld. Diese Veränderung im Feld kann das Lesegerät erkennen und decodieren. Somit erzeugt der RFID-Tag kein eigenes Feld, sondern beeinflusst nur das Feld vom Lesegerät.

Über RFID besteht bereits die Möglichkeit eine Authentifizierung durchzuführen. Ein RFID-Lesegerät mit der entsprechenden Software kann einen Tag authentifizieren. Wenn es über alle Schlüssel verfügt, kann es auch die Identität des Tags und damit die des Fahrzeugs feststellen.

3.3.2 RFID Authentifikation

Es ist ein Lesegerät und ein NXP-RFID-Tag vorhanden. Jeder Tag hat einen elektronischen Produktcode, den sogenannten "Electronic Product Code", kurz EPC.

Der EPC ist gleich zu setzen mit dem Namen eines Tags. Über den EPC kann jeder Tag eindeutig vom Lesegerät angesprochen und identifiziert werden. Der EPC eines Tags kann von berechtigten Nutzern verändert werden.

Der digitale Schaltkreis auf dem Tag unterstützt das sogenannte NXP UCODE DNA[10]. Die UCODE DNA ermöglicht kryptografische Authentifizierung und besitzt eine AES-128[11] Bit Verschlüsselung. Zudem erlaubt es die UCODE DNA den EPC zu verstecken. AES-128 ist ein symmetrisches Verschlüsselungsverfahren, sodass das Lesegerät die Schlüssel für die Authentifizierung kennen muss. Diese sind in einer Datenbank auf einer sogenannten "Trusted Plattform" gespeichert. Im einfachsten Fall ist die Trusted Plattform als Software Lösung realisiert. Deutlich sicherer ist eine dedizierte Hardware. Letztere kann entweder ein TPM Modul oder ein SE (NXP Secure Element)[12] sein.
Somit ist sichergestellt, dass nur die Software vom Lesegerät auf diese Schlüssel zugreifen kann.

[10] Kryptographische Funktion für den RFID UHF-Bereich [35].
[11] Advanced Encryption Standard mit 128 Bit Schlüssellänge [44].
[12] Security Mikrocontroller von NXP Semiconductors [34].

Die Tags werden in Gruppen unterteilt. Der EPC jedes Tags wird in einen Gruppen-EPC und einen Individuell-EPC unterteilt. Tags derselben Gruppe besitzen denselben Gruppen-EPC. Der EPC wird soweit versteckt, dass ein Lesegerät nur den Gruppen-Teil des EPC auslesen kann.

Ein Lesegerät kann somit beim ersten Lesezugriff nur den gekürzten EPC auslesen und erfährt nur zur welcher Gruppe dieser Tag gehört. Somit kann die genaue Identität des Tags nicht ohne Zugriffsberechtigung ausgelesen werden.

Zu jeder Gruppe gibt es einen zugehörigen Gruppenschlüssel, mit dem das Lesegerät auf den Speicher des Tags zugreifen kann. Zudem besitzt jeder Tag einen individuellen Privatschlüssel. Mit diesem Schlüssel kann sich der Tag authentifizieren und nachweisen, dass er wirklich der Tag ist.

3.3.3 Beispiel

Im nachfolgenden Beispiel wird erklärt, wie die Authentifizierung abläuft:
Wenn sich ein Tag im Lesebereich des Lesegeräts befindet, wird versucht, den EPC des Tags auszulesen. Die Software erkennt, dass es sich um einen gekürzten EPC handelt und startet eine Authentifizierung. Der gekürzte ausgelesene EPC wird mit der Datenbank verglichen, um festzustellen zu welcher Gruppe dieser Tag gehört. Wenn die Gruppe in der Datenbank registriert ist, wird daraus der Gruppenschlüssel ausgelesen.

Das Lesegerät sendet eine Zufallszahl, auch Random Challenge genannt, mit einem Authentifizierungsbefehl an den Tag. Bei diesem Befehl wird der Tag aufgefordert, der empfangenen Challenge seinen vollständigen EPC anzuhängen und diese mit dem Gruppenschlüssel zu verschlüsseln. Die verschlüsselte Challenge sendet der Tag an das Lesegerät.

Das Lesegerät kann nun diese Nachricht mit dem Gruppenschlüssel entschlüsseln und weiß nun, um welchen Tag es sich handelt. Zudem weiß das Lesegerät, ob die Antwort tatsächlich von dem Tag kam, dem das Lesegerät die Challenge geschickt hat. Die Challenge wird zufällig zur Laufzeit generiert, sodass diese nicht vorher manipuliert werden kann.

Der vollständige EPC, welchen das Lesegerät nun kennt, wird mit der Datenbank verglichen. Ist dieser Tag in dieser Gruppe registriert, wird eine zweite Authentifizierung gestartet. Dies ist notwendig, da jeder Tag aus einer Gruppe dieses Paket senden könnte. Somit wird sichergestellt, dass der Tag welcher sich authentifizieren möchte auch tatsächlich dieser Tag ist.

Das Lesegerät sendet erneut eine Random Challenge an den Tag, mit der Aufforderung, diese mit dem Privatschlüssel des Tags zu verschlüsseln. Die Challenge wird vom Tag mit seinem privaten Schlüssel verschlüsselt und an das Lesegerät gesendet.

Das Lesegerät entschlüsselt die empfangenen Daten. War die Entschlüsselung erfolgreich und ist die Challenge dieselbe, welche vom Lesegerät gesendet wurde, handelt es sich um einen gültigen Tag und das Lesegerät hat die zugeordnete Identität ermittelt.

4 Analyse der Anforderungen

In diesem Kapitel werden die Anforderungen für die Entwicklung zusammengetragen. Dabei werden Anforderungen, welche sich aus dem iKoPA-Projekt und von Seiten NXP ergeben, sowie Anforderungen, welche für die spätere Standardisierung notwendig sind, berücksichtigt

4.1 Anforderungen in iKoPA

In iKoPA ist je ein Use Case vorgesehen, indem ein Fahrzeug sich alternativ über V2X oder über RFID gegenüber einem Parkhaus authentifiziert. Diese Bachelorarbeit erarbeitet eine Lösung für die V2X Option, die nun im Folgenden vorgestellt werden soll. Die konzeptionelle Vorgehensweise wurde mit dem iKoPA Konsortium abgestimmt und durch die Bachelorarbeit ausgearbeitet und technisch umgesetzt.
Bei erfolgreicher Authentifizierung und vorheriger Reservierung wird dem Fahrzeug Zugang zu dem Parkhaus gewährt. Aus diesem Use Case lässt sich folgendes Modell ableiten.
In Abbildung 4.1 ist der Ablauf dargestellt. Der Nutzer registriert sich zunächst beim

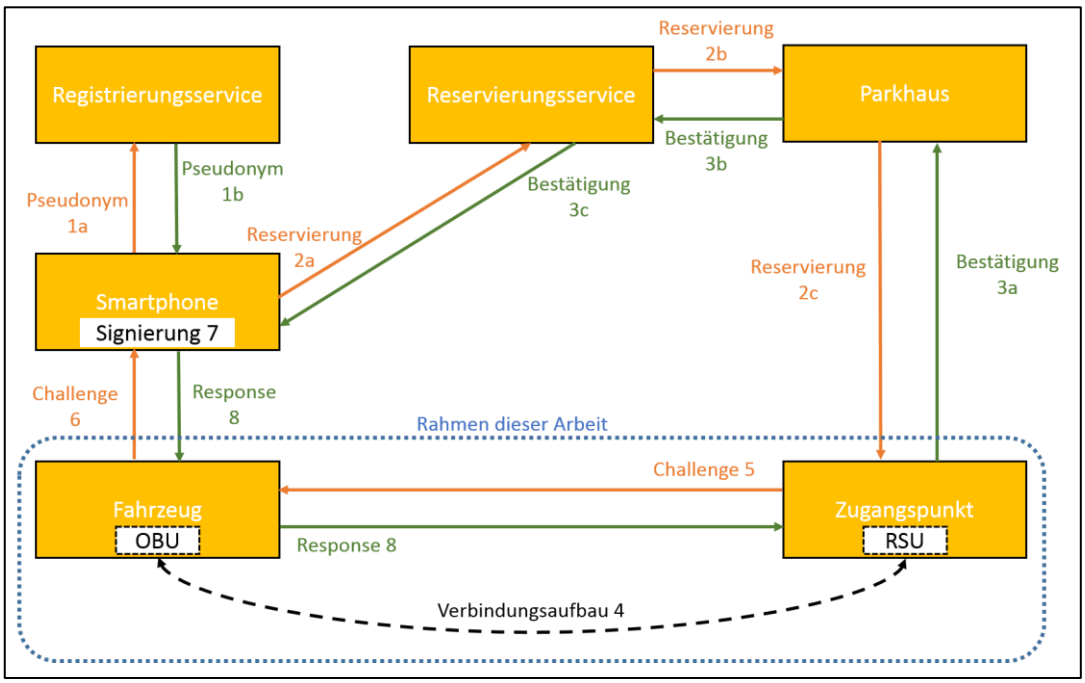

Abbildung 4.1 - Model Use Case Zugangskontrolle über V2X [25]

Registrierungsservice und erhält eine Pseudonym (1). Dieses Pseudonym beinhaltet ein Schlüsselpaar zur Authentifizierung. Nach erfolgreicher Registrierung kann der Benutzer unter Verwendung des Pseudonyms eine Reservierung tätigen, ohne seine Identität preisgeben zu müssen. Dies geschieht über den Reservierungsservice (2).
Der Reservierungsservice gibt die Anfrage an das Parkhaus weiter und erhält eine Bestätigung (3). Diese wird an das Smartphone weitergereicht. Nun fährt der Benutzer mit seinem Fahrzeug zum Parkhaus. Das Fahrzeug erkennt die Schranke des Parkhauses und startet eine Authentifizierung (4). Von der RSU der Schranke erhält die OBU im Fahrzeug eine Challenge mit der Aufforderung diese mit dem Ticket zu signieren (5).

Die OBU leitet die Challenge an das Smartphone weiter (6) und dieses signiert sie (7). Durch diesen Vorgang kann der Benutzter nachweisen, dass er die Buchung durchgeführt hat. Das Fahrzeug leitet über die OBU die signierte Challenge an die Schranke weiter (8). Bei erfolgreicher Authentifizierung erhält das Fahrzeug Zugang zum Parkhaus. Diese Arbeit konzentriert sich auf die Authentifizierung zwischen OBU und RSU. Die Signatur soll in iKoPA vom Smartphone erstellt werden. Da zum jetzigen Zeitpunkt die Smartphone Komponente für iKoPA noch nicht fertiggestellt ist, wird die Signaturerstellung von der OBU übernommen. Funktional bedeutet dies keine Einschränkung. Das Smartphone hat lediglich den Vorteil, dass der Fahrer die Signatur auch außerhalb des Fahrzeuges generieren kann. Realisierung direkt in der OBU hat den Vorteil leichterer Portierung, d.h., dass auch Authentifizierung in Anwendungen unabhängig von iKoPA leichter möglich ist. Aufbauend hierauf kann die Signaturerstellung später für iKoPA auf ein Smartphone ausgelagert werden. Aufbauend hierdrauf kann die Signaturerstellung später für iKoPA auf ein Smartphone ausgelagert werden.

Aus diesem Use Case lassen sich folgende Anforderungen, für eine Authentifizierung über V2X, ableiten.

1. Die Identität der Nutzer soll geheim bleiben. Es soll möglich sein, dass sich das Fahrzeug bei der Schranke authentifizieren kann, ohne seine Identität preiszugeben (Pseudonymisierung).
2. Die Schranke soll die Zugangsberechtigung prüfen können, ohne die Identität des Nutzers verfolgen zu können (Verkettbarkeit).
3. Das Authentifizierungsverfahren soll gegen unbefugten Zugriff geschützt werden. (Sicherheit im Sinne von *Security*).

4.2 Anforderung für Standardisierung

Das zu entwickelnde Kommunikationsprotokoll und die dafür benötigten Nachrichten sollen im Ausblick auf dieser Arbeit standardisiert werden.

Es soll die Möglichkeit gegeben werden, dass die Nachricht ähnlich der CAM und DENM Nachricht von jedem V2X-fähigen Gerät gesendet und empfangen werden kann.

Daraus lassen sich folgende Anforderungen für die Standardisierung ableiten:

4. Die Nachrichten sollten in ASN.1 definiert werden und einen ähnlichen Aufbau wie die CAM und DENM Nachricht haben.
5. Die Nachricht soll modular aufgebaut werden und erweiterbar sein.

4.3 Anforderungen von NXP

Für den in iKoPA beschriebenen Use Case soll ein Demonstrator von NXP aufgebaut werden. Dieser soll die Authentifizierung über V2X und RFID zeigen. Das Ergebnis der Authentifizierung soll optisch dargestellt werden.

Im Rahmen dieser Arbeit soll daher ein Demonstrator, bestehend aus einem Zugangspunkt und einem Fahrzeug, konzipiert, aufgebaut und in Betrieb genommen werden. Von NXP werden hierfür zwei OBUs von Cohda Wireless, sowie gehöriges Zubehör zur Verfügung gestellt.

Vom Projektpartner SWARCO wird für diesen Zweck ein optischer Signalgeber zur Verfügung gestellt [Abbildung 4.2].

Dieser verfügt über drei LEDs in den Farben Grün, Orange und Rot und wird über 12V Gleichspannung betrieben.

Abbildung 4.2 - Optischer Signalgeber Swarco [40]

Die Cohda Wireless MK5 verfügt über keine GPIOs zur Ansteuerung des optischen Signalgebers. Steuersignale müssen daher über ein Netzwerk generiert und angebunden werden.
Daraus ergeben sich noch folgende Anforderungen:

6. Es soll ein Demonstrator für NXP aufgebaut werden.
7. Für den Demonstrator sollen die von NXP zur Verfügung gestellten Komponenten genutzt werden.
8. Der Aufbau soll kompakt gehalten werden, transportierbar und einfach installierbar sein.

4.4 Anforderungsliste

Aus den voran gegangen Kapiteln lässt sich folgende Anforderungsliste erstellen.

Tabelle 4.1 - Anforderungsliste

	ANFORDERUNG	BESCHREIBUNG
1.	**Anonyme Identität**	Die Identität des Benutzers soll nicht preisgegeben werden.
2.	**Zugangskontrolle**	Der Kommunikationspartner soll die Berechtigung für eine Aktion prüfen können.
3.	**Sicherheit**	Kommunikation soll vor Angriffen von unbefugten geschützt werden.
4.	**ASN.1 Nachricht**	Die für die Kommunikation nötigen Nachrichten sollen in ASN.1 definiert werden.
5.	**Modularer Aufbau**	Die Nachricht, sowie die Software sollen Modular aufgebaut werden und einfach erweiterbar sein.
6.	**Demoaufbau**	Der Demonstrator soll aufgebaut werden.
7.	**Cohda Wireless MK5**	Für Implementierung soll die MK5 von Cohda Wireless verwendet werden.
8.	**Kompakter, portabler Aufbau**	Der Demonstrator muss kompakt transportierbar und einfach installierbar sein, da er Teil einer Projektdemo sein wird.

5 Konzeption

Auf Basis der herausgearbeiteten Anforderungen werden die Systemkomponenten identifiziert. Dabei wird zwischen Hardware- und Softwarekomponenten unterschieden. Aus den Anforderungen geht hervor, dass zwei Systeme miteinander kommunizieren. Auf der einen Seite ist das Fahrzeug mit dem Benutzer und der OBU, auf der anderen Seite befindet sich der Zugangspunkt mit einer RSU und einem optischen Signalgeber. Das Fahrzeug soll sich gegenüber dem Zugangspunkt authentifizieren, um Zugang zu erhalten [Abbildung 5.1].

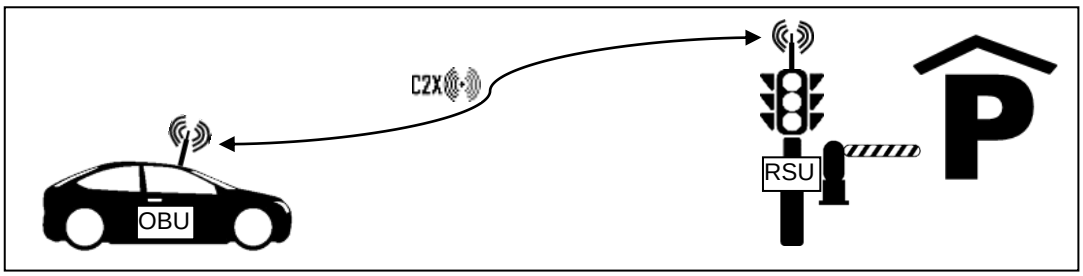

Abbildung 5.1 - Zugangskontrolle [25]

In diesem Fall authentifiziert sich nur das Fahrzeug gegenüber dem Zugangspunkt und vertraut diesem von seiner Seite aus. Es ist jedoch möglich, dass sich ein Angreifer als Schranke ausgibt, um das Fahrzeug in einen Hinterhalt zu locken (man denke an voll automatisierte Fahrzeuge). Daher wäre eine bidirektionale Authentifizierung als Erweiterung möglich und sinnvoll. Das Fahrzeug würde nach erfolgreicher Authentifizierung am Zugangspunkt eine Authentifizierung von der Schranke verlangen. Diese Schranke müsste beweisen, dass sie tatsächlich die Schranke ist, die sie ausgibt zu sein. Dabei kann dasselbe Verfahren genutzt werden wie bei der unidirektionalen Authentifizierung. Bei einer bidirektionalen Authentifizierung sind weiter Use Cases denkbar. So wäre auch eine gegenseitige Authentifizierung von zwei Fahrzeugen möglich. Diese könnten einander nach der Authentifizierung vertrauen und ihre Identitäten preisgeben oder geheime Daten austauschen [Abbildung 5.2].

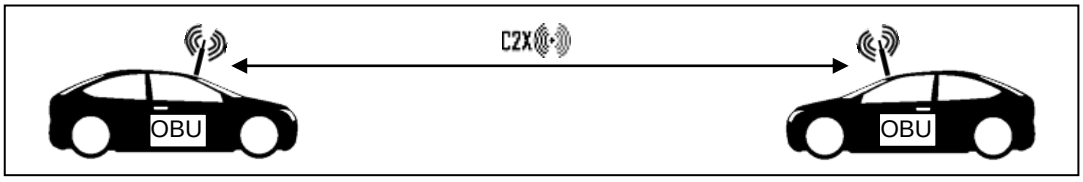

Abbildung 5.2 - Bidirektionale Authentifizierung [25]

5.1 Kommunikationsprotokoll

Für die Authentifizierung über V2X wird das in Kapitel 2.7.4 beschriebene Challenge-Response-Verfahren mit digitaler Signatur genutzt.

Die Challenge wird vom Empfänger signiert und dem Sender als Response zurückgeschickt. Dieser überprüft die Signatur. Dieses Verfahren ist für diesen Anwendungsfall sehr gut geeignet, da vor der Authentifizierung lediglich der öffentliche Schlüssel des Fahrzeuges an die Schranke übermittelt werden muss. Dies kann bei der Reservierung eines Parkplatzes geschehen, damit der Zugangspunkt weiß, dass nur das Fahrzeug mit dem dazu passenden privaten Schlüssel Zugang erhalten darf.

Damit weicht dieses Konzept bewusst von dem RFID-Konzept ab. Beim RFID-Konzept wurde, bedingt durch die RFID-Tags, symmetrische Kryptographie verwendet. Bei dem V2X-Konzept wird eine leistungsfähigere Hardware verwendet. Diese ermöglicht asymmetrische Kryptographie. Dadurch kann ein Signaturverfahren für die Authentifizierung genutzt werden. Beim der Authentifizierung über RFID wird einer Challenge die eigene Identität hinzugefügt und mit einem Schlüssel verschlüsselt. Durch den Besitz des Schlüssels wird die Identität nachgewiesen.

Bei der Authentifizierung über V2X wird die Identität nicht preisgegeben. Es wird lediglich bewiesen, dass die zuvor getätigte Reservierung von diesem Nutzer erfolgt ist.

Es wird eine Nachricht nach ASN.1 definiert, welche für das Versenden von Challenge und Response, sowie für weitere Befehle genutzt wird. Der Nachrichtenstruktur wird zusätzlich eine Zahl hinzugefügt. Diese Zahl repräsentiert den Inhalt der Nachricht, somit weiß der Kommunikationspartner, um welche Daten es sich handelt. In dieser Arbeit wird diese Zahl als „Operation Code“, kurz Opcode, bezeichnet, da sie Befehle für die nächste Operation definiert.

5.1.1 Ablauf

Es wird ein Schlüsselpaar, bestehend aus öffentlichem und privatem Schlüssel, für das Fahrzeug und ein Schlüsselpaar für den Zugangspunkt erzeugt.
Diese Schlüsselpaare werden für den Demoaufbau in dieser Arbeit lokal in den MK5 gespeichert, da noch keine Registrierungs- und Reservierungsservice, welche für die Schlüsselverwaltung zuständig sind, vorhanden sind.

In Abbildung 5.3 ist der Authentifizierungsablauf dargestellt. Im Folgenden wird dieser näher erklärt.

1. Benutzer startet Authentifizierungsvorgang.
2. OBU sendet einen Startbefehl an die RSU.
3. RSU generiert eine Zufallszahl r_B.
4. Zufallszahl r_B wird als Challenge an die OBU gesendet.
5. OBU generiert eine Zufallszahl r_A.
6. Die Antwort wird zusammengefügt aus der eigenen generierten Zufallszahl r_A, der empfangenden Zufallszahl r_B und der ID der Ziel-RSU.
7. Ein Hashwert wird aus zusammengefügten Zahlen berechnet.
8. Der Hashwert wird mit dem privaten Schlüssel der OBU signiert.
9. Der signierte Hashwert, die Zufallszahl r_A und die ID der Ziel-RSU werden zur Response hinzugefügt.
10. Die Response wird an die RSU gesendet.
11. Die Response wird dekodiert und auf die Zufallszahl r_A, die ID und den signierten Hashwert aufgeteilt.
12. RSU berechnet aus den Zufallszahlen r_A und r_B und seiner ID einen Hashwert.
13. Signatur wird mit dem eigens berechneten Hashwert und dem öffentlichen Schlüssel der OBU überprüft.

Signaturprüfung erfolgreich:

14. Die RSU bestätigt der OBU die erfolgreiche Authentifizierung.
15. Die RSU schickt den Befehl der erfolgreichen Authentifizierung an den optischen Signalgeber. Dieser schaltet die grüne Lampe an.
16. Dem Benutzer wird über die grüne Lampe eine erfolgreiche Authentifizierung signalisiert.

Signaturprüfung fehlgeschlagen:

14. Die RSU signalisiert der OBU die fehlgeschlagene Authentifizierung
15. Die RSU schickt den Befehl der fehlgeschlagenen Authentifizierung an den optischen Signalgeber. Dieser schaltet die rote Lampe an.
16. Dem Benutzer wird über die rote Lampe eine fehlgeschlagene Authentifizierung signalisiert.

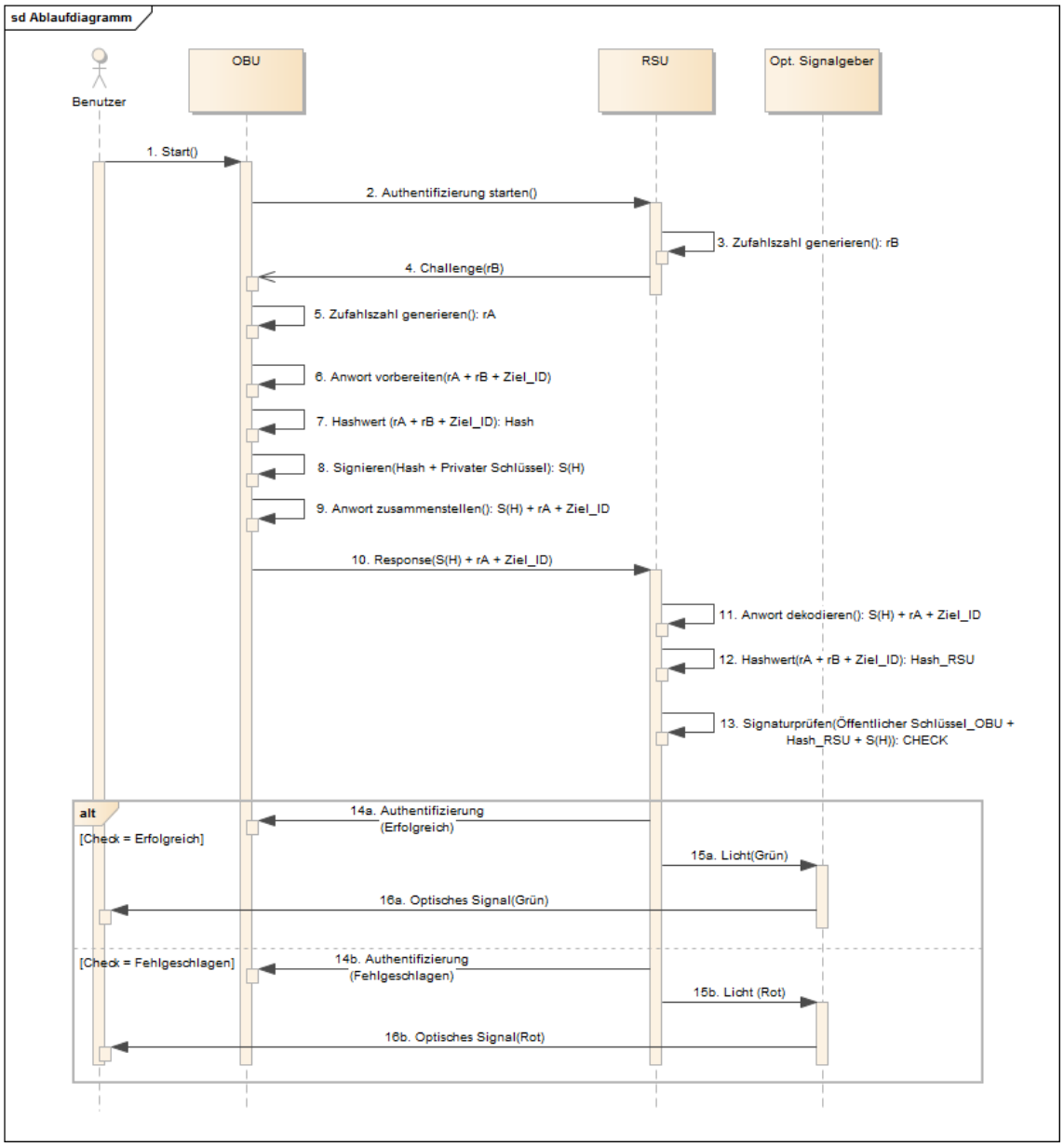

Abbildung 5.3 - Sequenzdiagramm [24]

5.1.2 Auth-Nachricht

Nach ASN.1 Standard wird eine Nachrichtstruktur definiert, welche für das Senden von Challenge und Response genutzt wird. Die Struktur der Nachricht wird ähnlich der Nachrichtenstruktur einer CAM-Nachricht [Code 2.] aufgebaut. Die Nachricht wird Auth-Nachricht genannt, da sie für eine Authentifizierung genutzt wird.

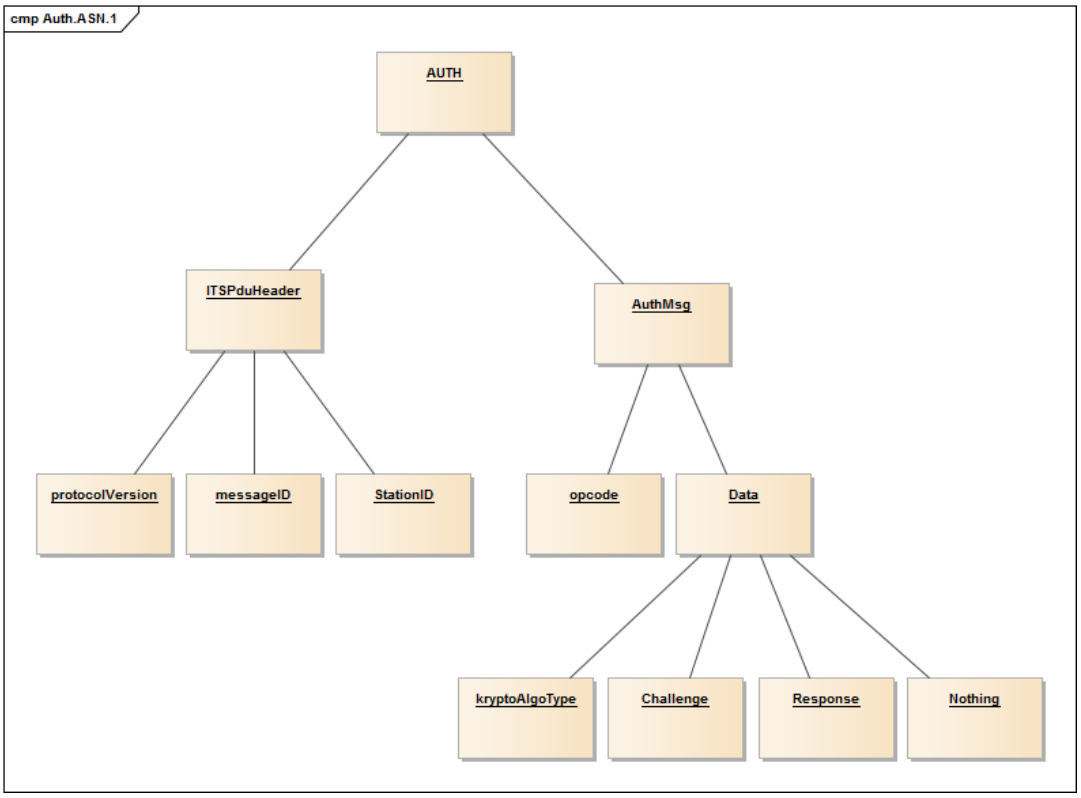

Abbildung 5.4 - Aufbau Auth-Nachricht [24]

In Abbildung 5.4 ist der Aufbau einer Auth-Nachricht in einem Baumdiagramm dargestellt.

Man sieht, dass die Auth-Nachricht, ähnlich einer CAM-Nachricht auf oberster Schicht aus zwei Blöcken besteht. Zu einem, dem ITSPduHeader mit der protocolVersion, der messageID und der StationID, zum anderen aus der AuthMsg mit dem Opcode und dem Data Block.

In dem Data Block sind die vier Blöcke, cryptoAlgoType, Challenge, Response und Nothing definiert. Diese können von der Software mit Inhalt befüllt werden.

5.1.3 Operation Codes

Für die Authentifizierung über ein Challenge-Response-Verfahren müssen verschiedene Daten ausgetauscht werden. Damit der Kommunikationspartner weiß, um welche Daten es sich handelt, wird eine Zahl der Nachricht hinzugefügt, der sogenannte Opcode, welcher verschiedene Befehle definiert.

In dieser Arbeit wurden folgende Opcodes definiert und für die Kommunikation genutzt.

Tabelle 5.1 - Opcodes

OPCODE	BEZEICHNUNG	BEDEUTUNG	DATEN
0	Noopcode	Kein Inhalt in dieser Nachricht	Keine Daten
1	ReadyForAuth	Bereit für eine Authentifizierung	Keine Daten
2	StartAuth	Starte eine einseitige Authentifizierung	Verwendeter Verschlüsselungsalgorithmus
3	StartBiAuth	Starte eine gegenseitige Authentifizierung	Verwendeter Verschlüsselungsalgorithmus
4	ChallengeForSign	Challenge zum Signieren	Challenge
5	SignResponse	Signierte Antwort	Response
6	AuthSuccessful	Authentifizierung erfolgreich	Keine Daten
7	AuthFailed	Authentifizierung fehlgeschlagen	Keine Daten
8	AccessGrant	Zugang gewährt	Keine Daten
9	AccessDenied	Zugang verweigert	Keine Daten

5.2 Hardwareaufbau

Der Hardwareaufbau ist in zwei Bereiche aufgeteilt. Auf der einen Seite befindet sich ein Fahrzeug, auf der anderen ein Zugangspunkt [Abbildung 5.5].

Im Rahmen dieser Arbeit stehen weder ein richtiges Fahrzeug noch ein richtiger Zugangspunk zur Verfügung. Daher wird die Cohda MK5 OBU mit Peripheriegeräten als Fahrzeug betrachtet und in einem kleinen Gehäusemodell untergebracht.

Auf der Zugangspunktseite wird ein optischer Signalgeber mit einer Cohda MK5 OBU als Zugangspunkt betrachtet. Alternativ oder additiv kann hier künftig auch eine mechanische Schranke zu Einsatz kommen.

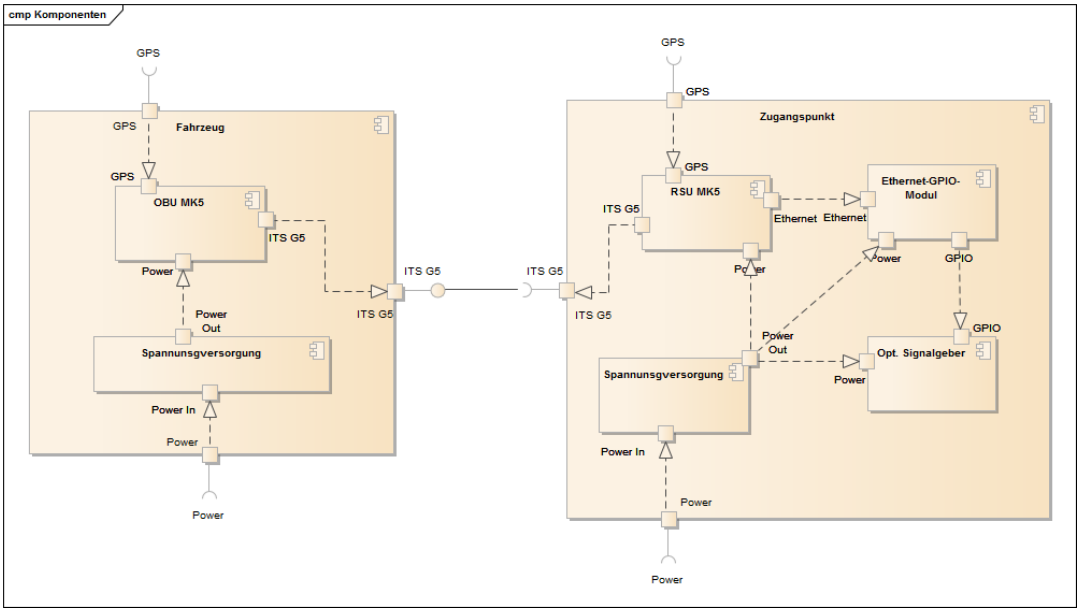

Abbildung 5.5 - Komponenten [24]

5.2.1 Fahrzeug

Zur benötigten Fahrzeughardware gehören eine OBU, 802.11p Antennen, eine GNSS-Antenne und eine dazugehörige Spannungsversorgung. Für die Entwicklung und zu Debug[13]-Zwecken wird ein Notebook genutzt. Für den Demonstrator wird ein Samsung Galaxy Tab A mit Android Betriebssystem genutzt.

Um eine einfache Kommunikation zwischen dem Notebook/Tablet und der Cohda MK5 aufzubauen, wird ein Ethernet-to-Wifi Router genutzt. Für diesen wird ein DC/DC Konverter benötigt, um die Spannung anzupassen.

Tabelle 5.2 - Komponentenliste Fahrzeug

	KOMPONENTE
OBU	Cohda Wireless MK5
GNSS/802.11P ANTENNE	Cohda Wireless
ETHERNET-TO-WIFI ROUTER	TP-Link TL-WR702N
SPANNUNGSVERSORGUNG +12V	VOLTCRAFT FTPS 12-27W 12 V/DC 2250 mA
SPANNUNGSVERSORGUNG +5V	Profi Power USB-Adapter
TABLET	Samsung Galaxy Tab A

In Tabelle 5.2 sind die ausgewählten Komponenten aufgelistet.

Als Router wird der TP-Link TL-WR702N genutzt. Dieser ist sehr klein und bietet die Funktionalität, als Ethernet-to-Wifi Router genutzt zu werden.

Als Spannungsversorgung wird ein Tischnetzteil von Voltcraft genutzt. Diese stellt 12V/ 2250 mA zur Verfügung, welches für den Betrieb der Cohda MK5 und des Ethernet-to-Wifi Router ausreicht. Für den Ethernet-to-Wifi Router wandelt ein DC/DC Konverter die +12V in +5V um.

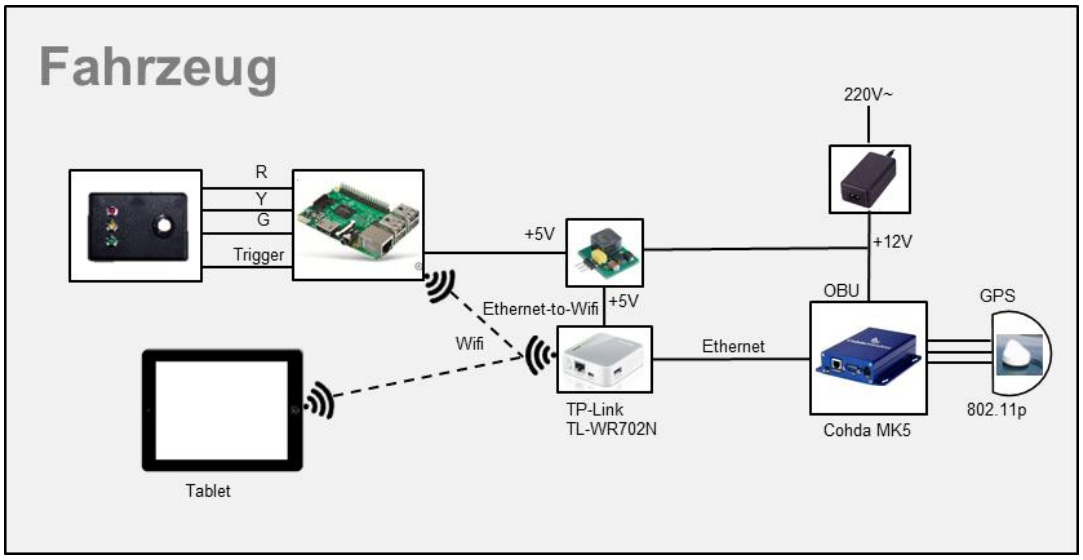

Abbildung 5.6 - Aufbau Komponenten Fahrzeug [25]

In Abbildung 5.6 sind schematisch die Komponenten für den Aufbau des Fahrzeugs und deren Signalflüsse dargestellt. Zum besseren Verständnis sind Bilder der einzelnen Komponenten in die Grafik eingearbeitet.

[13] Prozess zum Finden und Lösen von Fehlern [42]

Das Netzteil wandelt die 220 V Netzspannung in +12V um und versorgt die OBU, sowie den DC/DC Konverter, welcher wiederum den Ethernet-to-Wifi Router versorgt. Das Notebook baut eine Wifi-Verbindung zum Router auf und verbindet sich darüber mit der Cohda MK5. So kann die Software auf die Cohda MK5 geladen und gestartet werden. Über eine SSH-Verbindung ist es möglich, für Debugging-Zwecke Ausgaben auf der Konsole anzuzeigen.

5.2.2 Zugangspunkt

Der Zugangspunkt besteht, wie das Fahrzeug, auch aus einer OBU von Cohda Wireless, einer GNSS/802.11p Antenne, einer Spannungsversorgung und einem Ethernet-to-Wifi Router. Funktional sind OBU und RSU nahezu identisch; die RSU ist mechanisch robuster und für Außenmontage ausgelegt. Daher war die Verwendung einer OBU sinnvoll, da sie zu einem kompakteren Aufbau führt. Zudem verfügt der Zugangspunkt über einen optischen Signalgeber. Dieser wird von einem Einplatinencomputer mit GPIO-Relais angesteuert. Der Einplatinencomputer wird über eine TCP/IP Verbindung von der OBU gesteuert.

Tabelle 5.3 - Komponentenliste Zugangspunkt

	KOMPONENTE
OBU	Cohda Wireless MK5
GNSS/802.11P ANTENNE	Cohda Wireless
ETHERNET-TO-WIFI ROUTER	TP-Link TL-WR702N
SPANNUNGSVERSORGUNG +12V	VOLTCRAFT FTPS 12-27W 12 V/DC 2250 mA
SPANNUNGSVERSORGUNG +5V	Profi Power USB-Adapter
EINPLATINENCOMPUTER	Raspberry Pi 3
GPIO RELAIS	4-Kanal Relais Modul 12V/230V
OPTISCHER SIGNALGEBER	Swarco Futura 12VDC LED
RFID LESEGERÄT	M-ARU

In Tabelle 5.3 sind die genutzten Komponenten aufgelistet.

Als OBU wird auch hier die Cohda Wireless MK5 OBU genutzt. Auch der TP-Link TL-WR702N wird hier als Ethernet-to-Wifi Router genutzt.

Für die optische Darstellung wird der Signalgeber von Swarco genutzt. Die einzelnen Lampen werden über eine +12V Spannung eingeschaltet. Für das Ansteuern wird ein 4 Kanal Relais-Board genutzt. Dieses verfügt über vier Relais welche über GPIO-Ausgänge angesteuert werden können.

Da die Cohda MK5 OBU über keine freien GPIO-Ausgänge verfügt, wird eine zusätzliche Steuereinheit benötigt. Dafür eignet sich der Raspberry Pi 3 sehr gut. Dieser verfügt über eine Ethernet-Schnittstelle und eine Wifi-Schnittstelle. Zudem verfügt er über frei programmierbare GPIOs. Über eine TCP/IP Verbindung wird eine Verbindung von der Cohda MK5 zum Raspberry Pi aufgebaut. Dadurch ist eine seriell angebundene Steuerung der GPIOs möglich, bedurfte aber einer zusätzlichen Programmierarbeit, die ursprünglich nicht eingeplant war.

Eine weitere Möglichkeit wäre, ein Ethernet-to-GPIO Board zu nutzen. Diese ist jedoch nicht so flexibel wie ein freiprogrammierbarer Raspberry Pi. Zudem wird der Raspberry Pi 3 für die Ansteuerung des RFID-Readers verwendet, sodass dieser bereits vorhanden ist.

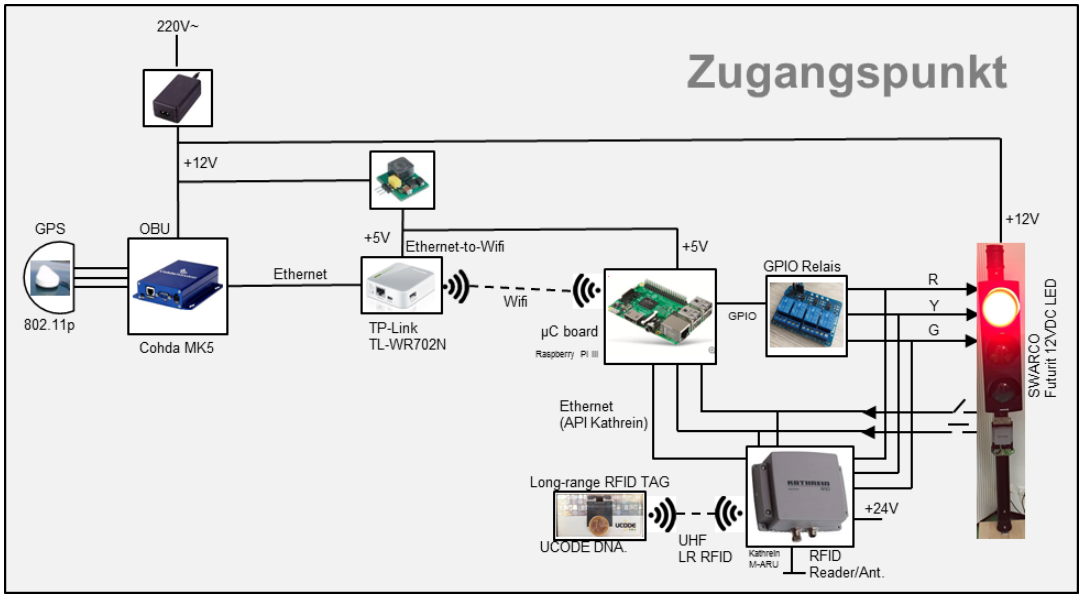

Abbildung 5.7 - Aufbau Komponenten Zugangspunkt [25]

In Abbildung 5.7 sind die Komponenten des Zugangspunktes schematisch dargestellt. Ähnlich wie beim Fahrzeug ist auch hier eine Cohda MK5 OBU für die Kommunikation über ITS G5 zuständig. Für die Ansteuerung des optischen Signalgebers von Swarco wird ein Raspberry Pi 3 mit einem GPIO-Relais Board verwendet. Der Signalgeber benötigt eine +12V Spannungsversorgung. Diese wird vom Netzteil genutzt, welches die OBU versorgt.

5.2.3 Gesamtsystem

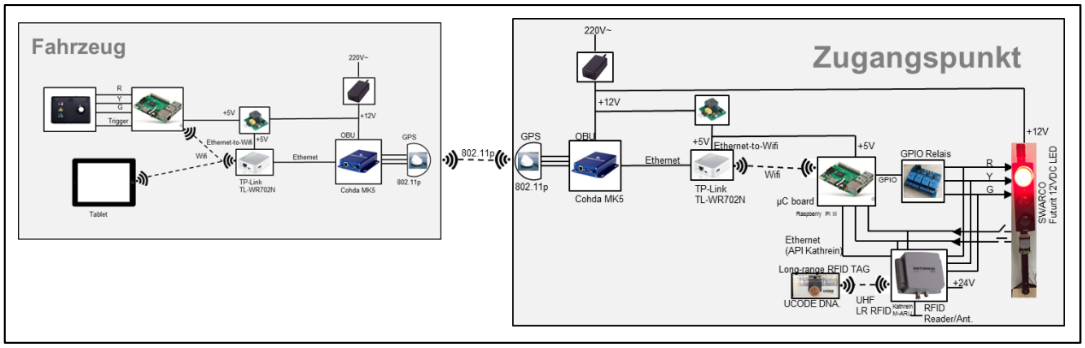

Abbildung 5.8 - Aufbau Gesamtsystem [25]

In Abbildung 5.8 ist der Aufbau des Gesamtsystems aufgezeigt. Dies besteht aus dem Fahrzeugaufbau und dem Zugangspunkt. Diese kommunizieren über 802.11p miteinander.

5.2.4 MK5 - GPIO Verbindung

Für die GPIO-Ansteuerung wird wie oben beschrieben ein Raspberry Pi 3 genutzt.

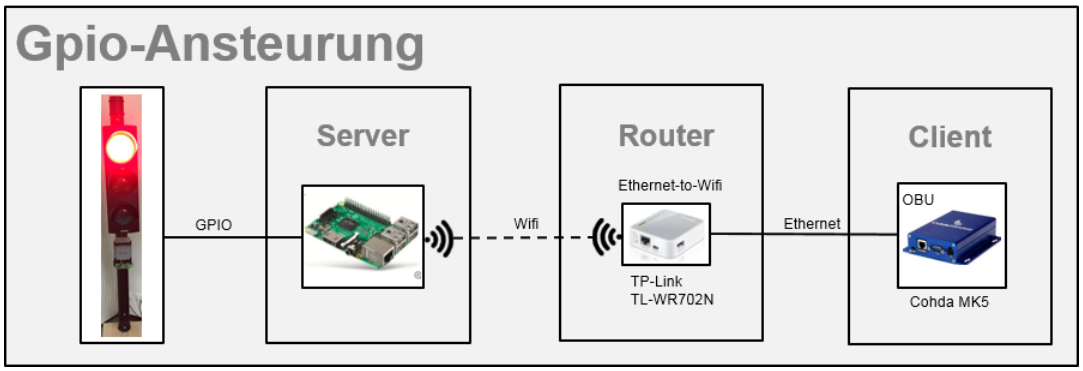

Abbildung 5.9 - Aufbau Gpio Ansteuerung [25]

In Abbildung 5.9 ist das Konzept der GPIO-Ansteuerung dargestellt.
Man erkennt das die OBU über den Wifi-Router und den Raspberry Pi den optischen Signalgeber schaltet.

Auf dem Raspberry Pi 3 wurde eine Server-Software implementiert. Diese baut über den Wifi-Router eine TCP/IP Verbindung zur Server-Software auf der Cohda MK5 auf.
Dafür müssen der Server, sowie der Client eine feste IP-Adresse vom Router zugewiesen bekommen.
In den Einstellungen des Routers, sowie in der Server- und Client-Software werden folgende IP-Adressen festgelegt.

- Raspberry Pi 3: 192.168.42.129
- Cohda MK5: 192.168.42.128

Die Client-Software wird mit in die Authentifizierungssoftware der MK5 implementiert. Beim Starten der Authentifizierungssoftware wird nach dem Server gesucht. Wenn dieser gefunden wurde, wird eine Verbindung aufgebaut.

Bei erfolgreicher Verbindung erfolgt ein Selbsttest. Dazu werden alle GPIOs kurz angesteuert.

5.3 Softwareaufbau

Für die Softwareentwicklung wird die Cohda SDK genutzt. In dieser sind mehrere Softwarebeispiele implementiert.

Eins davon ist das „exampleETSI“, welches als Grundlage zum Senden und Empfangen von Daten über ITS-G5 genutzt werden kann. Auf dieser Software aufbauend wird die Authentifizierungssoftware entwickelt. Der Quellcode wird in der SDK für die MK5 OBU kompiliert und übertragen. Über eine SSH-Verbindung wird die Software gestartet. Es sind verschiedene Konfigurationen für das „exampleETSI“ vorhanden. In dieser Arbeit wird nur die für diese Entwicklung benötigte Konfiguration beschrieben.

Als Konfiguration wird eine sogenannte „RAW ITS-Konfiguration“ genutzt. Diese ermöglicht das Senden und Empfangen von eigens definierten ASN.1 Nachrichten. Beim Starten der Software werden alle benötigten Konfigurationen zum Senden und Empfangen von Nachrichten geladen und konfiguriert. Es wird ein zyklischer Prozess (engl. Thread) gestartet welcher in einem bestimmten Zeitintervall eine Beispielnachricht sendet.

In Code 5.1 ist der Ausschnitt aus dem Code der exampleETSI dargestellt, welcher zyklisch eine Nachricht sendet.

```
1         // Thread loop
2         while ((pRAW->ThreadState & RAWITS_THREAD_STATE_STOP) == 0)
3         {
4
5               // polling delay
6               UtilNap(pRAW->Params.TxInterval, &Time);
7
8               // Send something
9               RAWIts_ReqSend(RAWHandle, &pRAW->Params)
10
11        }
```

Code 5.1 - zyklischer Prozess [Anhang: ThreadLoopOriginal.c]

In Code 5.1 – Zeile 2 wird eine while-Schleife aufgerufen, solange der Prozess nicht gestoppt wird. In dieser Schleife wird in Code 5.1 – Zeile 9 die Funktion „RAWIts_ReqSend()“ aufgerufen. Diese sendet eine Nachricht. Die Funktion „UtilNap()“ in Code 5.1 – Zeile 6 sorgt dafür, dass das Senden nur in einem bestimmten Zeitintervall geschieht.

Die für die Authentifizierung entwickelte Software wird in dieser while-Schleife implementiert, sodass der bereits vorhandene Softwarerahmen genutzt werden kann.

5.3.1 Zustände

Für die Authentifizierung über ITS G5 wird ein Kommunikationsprotokoll aufgebaut, welches auf dem Modell eines Zustandsautomaten basiert.
Bei einem Zustandsautomaten, auch Zustandsmaschine oder engl. Finite State Machine (FSM) genannt, handelt es sich um ein Modell, welches aus Zuständen, Zustandsübergängen und Aktionen besteht.

Für die Authentifizierung sind die in Tabelle 5.4 folgende Zustände notwendig. Die FSM wird so aufgebaut, dass sie sowohl als OBU, als auch als RSU betrieben werden kann.

Tabelle 5.4 - Zustände

ZUSTÄNDE	BESCHREIBUNG
Start	Initialisierung der Variablen und laden der Schlüssel.
WaitForAuth	Zugangspunkt wartet auf ein Fahrzeug, dass sich Authentifizieren möchte.
ReadyForAuth	Fahrzeug ist bereit für eine Authentifizierung und wartet auf Kontakt zum Ziel-Zugangspunkt.
StartAuth	Fahrzeug hat Kontakt zum Ziel-Zugangspunkt und gibt den Befehl zum Starten eine Authentifizierung.
WaitForChallenge	Fahrzeug wartet auf eine Challenge vom Zugangspunkt.
SendChallenge	Zugangspunkt generiert eine Challenge und sendet diese ans Fahrzeug.
WaitForResponse	Zugangspunkt wartet auf eine signierte Response
SendSignResponse	Fahrzeug signiert die empfangene Challenge und sendet diese zurück an den Zugangspunkt
WaitForDecision	Fahrzeug wartet auf eine Entscheidung des Zugangspunktes
CheckResponse	Zugangspunkt überprüft die signierte Response
AuthSuccessful	Zugangspunkt sendet, dass die Authentifizierung erfolgreich war.
AuthFailed	Zugangspunkt sendet, dass die Authentifizierung fehlgeschlagen ist.
WaitForAccess	Fahrzeug wartet auf Zugang.
AccessGrant	Zugangspunkt gibt den Zugang frei und sendet dies an das Fahrzeug.
AccessDenied	Zugangspunkt verweigert Zugang und sendet dies an das Fahrzeug.
End	Authentifizierung beendet.
Error	Fehler
TempError	Temporärer Fehler. System startet neu.

Die Authentifizierung könnte nach dem Zustand „CheckResponse“ zu Ende sein, wenn sich nur das Fahrzeug an der Schranke authentifizieren würde. Die Software ist so aufgebaut, dass auch eine gegenseitige Authentifizierung möglich ist.
Daher wird dem Kommunikationspartner der Status der Authentifizierung mitgeteilt. Erst zum Ende erfolgt die Freigabe des Zugangs bei erfolgreicher Authentifizierung.

Im weiteren Verlauf der Arbeit wird die einseitige Authentifizierung betrachtet. Die gegenseitige Authentifizierung ist optional entwickelt und wurde in das Konzept mit eingearbeitet.

5.3.2 Zustandsautomat – Gesamtes System

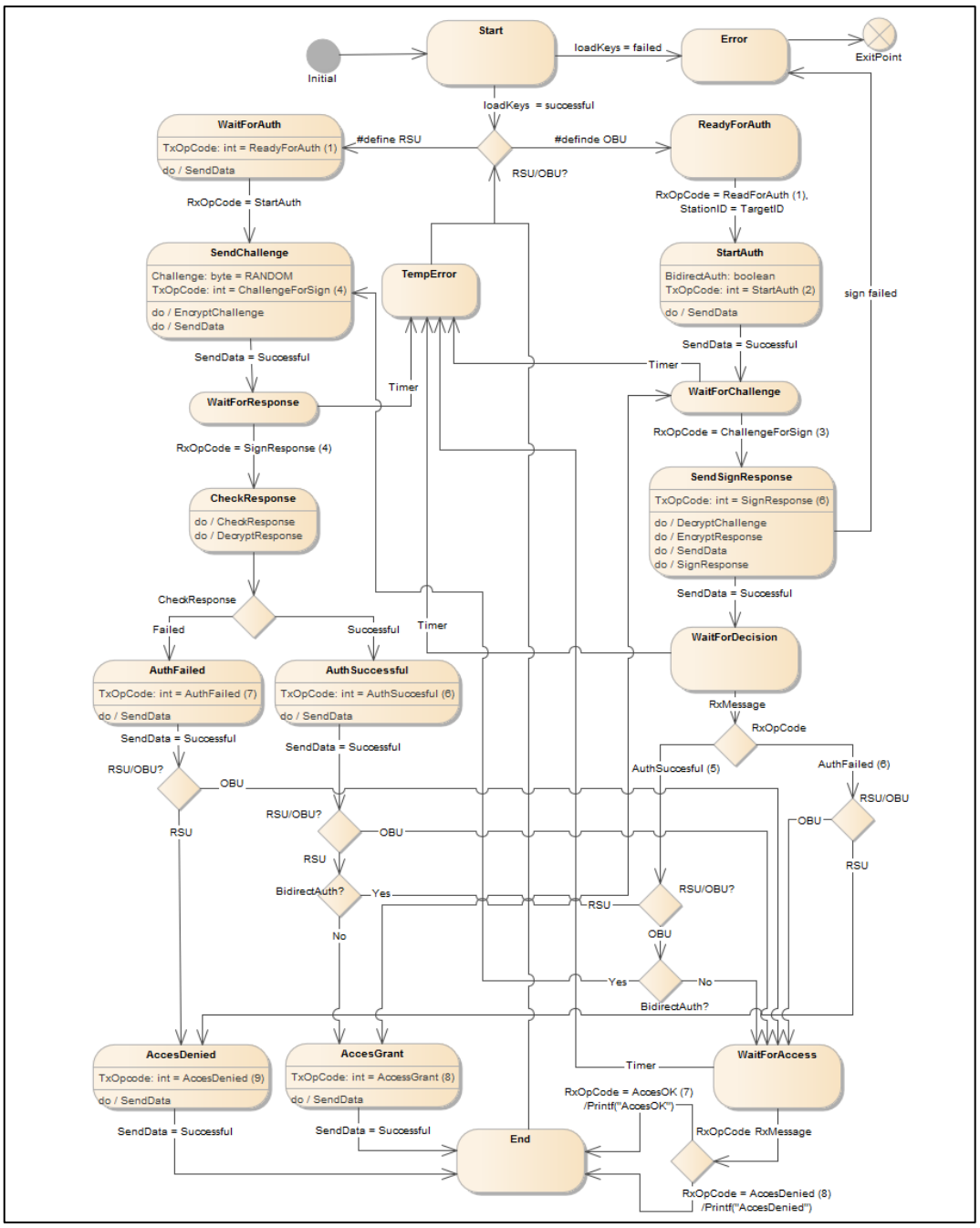

Abbildung 5.10 - Ablauf des Zustandsautomaten mit optional gegenseitiger Authentifizierung [24]

In Abbildung 5.10 ist das Zustandsdiagramm der gesamten Software dargestellt.
Nach dem initialen Start der Software wird der Zustand Start aufgerufen. In diesem werden Variablen initialisiert und Schlüssel für das Signieren und Verifizieren geladen. In dieser Arbeit werden die Schlüssel lokal abgespeichert. Später sollen diese von einem Server gemanagte werden. Für ein besseres Verständnis wird das Zustandsdiagramm im weiteren Verlauf aufgeteilt, auf ein Diagramm aus der Sicht der OBU [Abbildung 5.11] und eins aus der RSU [Abbildung 5.12] Sicht. Da das Gesamtsystem aus Sicht des Zustandsautomaten OBU beschrieben wird, ergibt sich aus dieser Beschreibung auch das Verhalten der Gegenseite, also der RSU. Daher wird auf eine detaillierte Beschreibung der RSU verzichtet.

5.3.3 Zustandsautomat – OBU

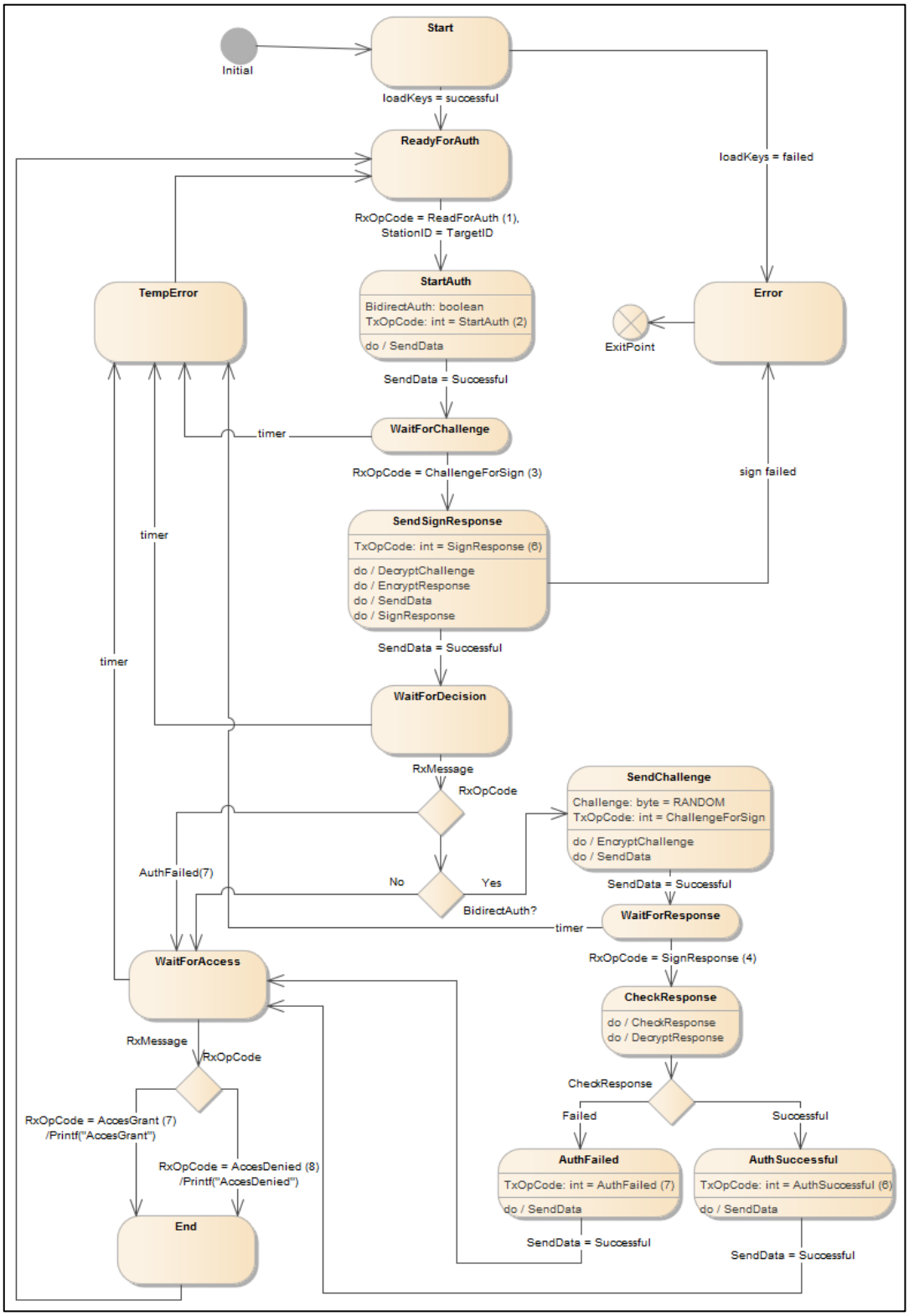

Abbildung 5.11 - Zustandsdiagramm OBU [24]

Start
Die OBU lädt die Schlüssel für das Signieren und Verifizieren aus einer Datei und speichert diese lokal ab. Der Zufallsgenerator wird initialisiert und alle Variablen werden auf einen Initialwert gesetzt.

Falls dies nicht erfolgreich war, wird in den Zustand „Error“ gewechselt und die Software wird beendet. Wenn die Initialisierung erfolgreich war, wird in den Zustand „ReadyForAuth“ gewechselt.

ReadyForAuth
Im Zustand „ReadyForAuth“ wird auf einen Kommunikationspartner gewartet.

Wenn ein Kommunikationspartner gefunden wurde, welcher den Opcode „ReadyForAuth (1)“ sendet und die Ziel-ID besitzt, wird in den Zustand „StartAuth“ gewechselt.

Für den Demonstrator wurde eine zusätzliche Bedingung implementiert. Die Auswertung des Zustandes „ReadyForAuth“ beginnt erst, wenn ein Triggersignal erkannt wurde. Diese Triggersignal wird im derzeitigen Demonstrator in Form eines Tasters implementiert. iKoPA wird hier voraussichtlich eine Kontaktschleife oder eine Lichtschranke vorsehen.

StartAuth
In diesem Zustand wird eine Nachricht mit dem Opcode „StartAuth (2)“ gesendet. Damit signalisiert das Fahrzeug, dass es sich bei dem Zugangspunkt authentifizieren möchte.
Es kann auch der Opcode „StartBiAuth (3)“ gesendet werden.
Dieser sagt dem Zugangspunkt, dass eine gegenseitige Authentifizierung gestartet werden soll und dieser sich auch bei dem Fahrzeug authentifizieren soll.
Den beiden Opcodes, zum Starten einer Authentifizierung, wird als Daten die Art des Verschlüsselungsalgorithmus hinzugefügt.

Wenn das Senden erfolgreich war wird in den Zustand „WaitForChallenge“ gewechselt, falls nicht wird die Nachricht erneut gesendet.

WaitForChallenge
Im Zustand „WaitForChallenge“ wird keine Aktion ausgeführt.

Es wird auf eine Nachricht mit dem Opcode „ChallengeForSign (4)“ gewartet. Wenn diese empfangen wurde wird in den Zustand „SendSignResponse“ übergegangen. Falls innerhalb einer bestimmten Zeit keine Nachricht mit „ChallengeForSign“ empfangen wird, wird davon ausgegangen, dass die Nachricht verpasst worden ist oder die Verbindung abgebrochen ist. Es wird dann in den Zustand „TempError“ übergegangen.

SendSignResponse
Im Zustand „SendSignResponse“ wird die Challenge nach dem in 0 beschriebenen Challenge-Response-Verfahren signiert. Wenn im Zustand „StartAuth“ eine Verschlüsselung vorgegeben worden ist, wird die Response nach diesem Verschlüsselungsverfahren verschlüsselt. Wenn keine Verschlüsselung vorgegeben worden ist, wird die Response unverschlüsselt in einer Nachricht mit dem Opcode „SignResponse (5)“ gesendet.

Nach erfolgreichem Senden, wird in den Zustand „WaitForDecision“ gewechselt.

WaitForDecision
Im Zustand „WaitForDecision“ wird auf eine Nachricht mit dem Opcode „AuthSuccessful (6)“ oder „AuthFailed (7)“ gewartet. Damit wird der OBU mitgeteilt, ob die Authentifizierung erfolgreich war oder fehlgeschlagen ist.

Wenn nach einer bestimmten Zeit keine Antwort empfangen wird, wird die Authentifizierung abgebrochen und es wird in den Zustand „TempError“ gewechselt.
Wenn der Opcode „AuthSuccessful (6)“ empfangen wird, wird überprüft ob es eine gegenseitige oder eine einseitige Authentifizierung ist.
Bei einer einseitigen Authentifizierung wird in den Zustand „WaitForAccess“ gewechselt.
Bei einer gegenseitigen Authentifizierung wird in den Zustand „SendChallenge“ gewechselt.

SendChallenge
Im Zustand „SendChallenge“ wird eine Zufallszahl generiert und als Challenge an den Kommunikationspartner gesendet. Die Nachricht wird mit dem Opcode „ChallengeForSign (4)“ versehen.

Nach dem Senden der Challenge wird in den Zustand „WaitForResponse“ gewechselt.

WaitForResponse
Hier wird nun auf eine Nachricht mit dem Opcode „SignResponse (5)“ gewartet. Mit dieser Nachricht wird die signierte Antwort empfangen.

Bei empfangen dieser Nachricht wird in den Zustand „CheckResponse“ gewechselt.

CheckResponse
Im Zustand „CheckResponse“ wird die empfangene Antwort aufgeschlüsselt und die darin enthaltende Signatur überprüft.

Ist die Signatur richtig wird in den Zustand „AuthSuccessful“ gewechselt. Sollte die Signatur falsch sein wird in den Zustand „AuthFailed“ gewechselt.

AuthSuccessful
Hier wird eine Nachricht mit dem Opcode „AuthSuccessful (6)“ an den Kommunikationspartner geschickt. Dadurch weiß dieser, dass die Authentifizierung erfolgreich war.

Nach dem Senden wird in den Zustand „WaitForAccess“ gewechselt.

AuthFailed
Im Zustand AuthFailed wird eine Nachricht mit dem Opcode „AuthFailed (7)“ an den Kommunikationspartner geschickt. Dadurch weiß dieser, dass die Authentifizierung fehlgeschlagen ist.

Nach dem Senden wird in den Zustand „WaitForAccess“ gewechselt.

WaitForAccess
Im Zustand WaitForAccess wird auf eine Nachricht mit dem Opcode „AccessGrant (8)“ oder „AccessDenied (9)“ gewartet. Dadurch wird dem Fahrzeug mitgeteilt, ob es Zugang erhält oder, ob dieser verweigert wird.

Nachdem diese Nachricht empfangen wurde wird in den Zustand „End“ gewechselt.

End
Hier wird die Authentifizierung für beendet erklärt.
Es wird in den Zustand „ReadyForAuth“ gewechselt.

TempError
In diesen Zustand wird gewechselt, wenn beim Warte auf eine Nachricht, die vorgegebene Wartezeit überschritten wird. Falls dies der Fall ist, wird davon ausgegangen, dass die Kommunikation abgebrochen worden ist.
In diesem Zustand werden, alle Variablen zurückgesetzt und es wird in den Zustand „ReadyForAuth“ gewechselt.

Error
Sollte bei Laden der Schlüssel oder beim Signieren ein Fehler auftreten, ist eine Authentifizierung nicht möglich.
Es wird in den Zustand Error gewechselt und die Software wird beendet. Der Fehler muss daraufhin behoben werden und die Software muss neu gestartet werden.

Im Folgenden werden die zusätzlichen Zustände in der RSU-Version erläutert. Alle anderen Zustände in der RSU-Version [Abbildung 5.12] sind gleich den Zuständen der OBU-Version.

WaitForAuth
In diesem Zustand wird periodisch eine Nachricht mit dem Opcode „ReadyForAuth (1)“ gesendet.

Dieser Zustand wird erst verlassen, wenn eine Nachricht mit dem Opcode „StartAuth (2)“ oder „StartBiAuth (3)“ empfangen wurde. Es wird abgespeichert, ob eine einseitige oder gegenseitige Authentifizierung gestartet wurde und welcher Verschlüsselungsalgorithmus genutzt wird. Danach wird in den Zustand „SendChallenge“ gewechselt.

AccessGrant
In diesem Zustand wird der Zugang freigegeben. An den Einplatinencomputer wird der Befehl zum Einschalten der grünen Lampe geschickt. Zudem wird eine Nachricht mit dem Opcode „AccessGrant (8)“ an das Fahrzeug gesendet.
Danach wird in den Zustand „End“ übergegangen.

AccessFailed
In diesem Zustand wird der Zugang nicht freigegeben. An den Einplatinencomputer wird der Befehl zum Einschalten der roten Lampe geschickt. Zudem wird eine Nachricht mit dem Opcode „AccessDenied (9)“ an das Fahrzeug gesendet.
Danach wird in den Zustand „End“ übergegangen.

5.3.4 Zustandsautomat – RSU

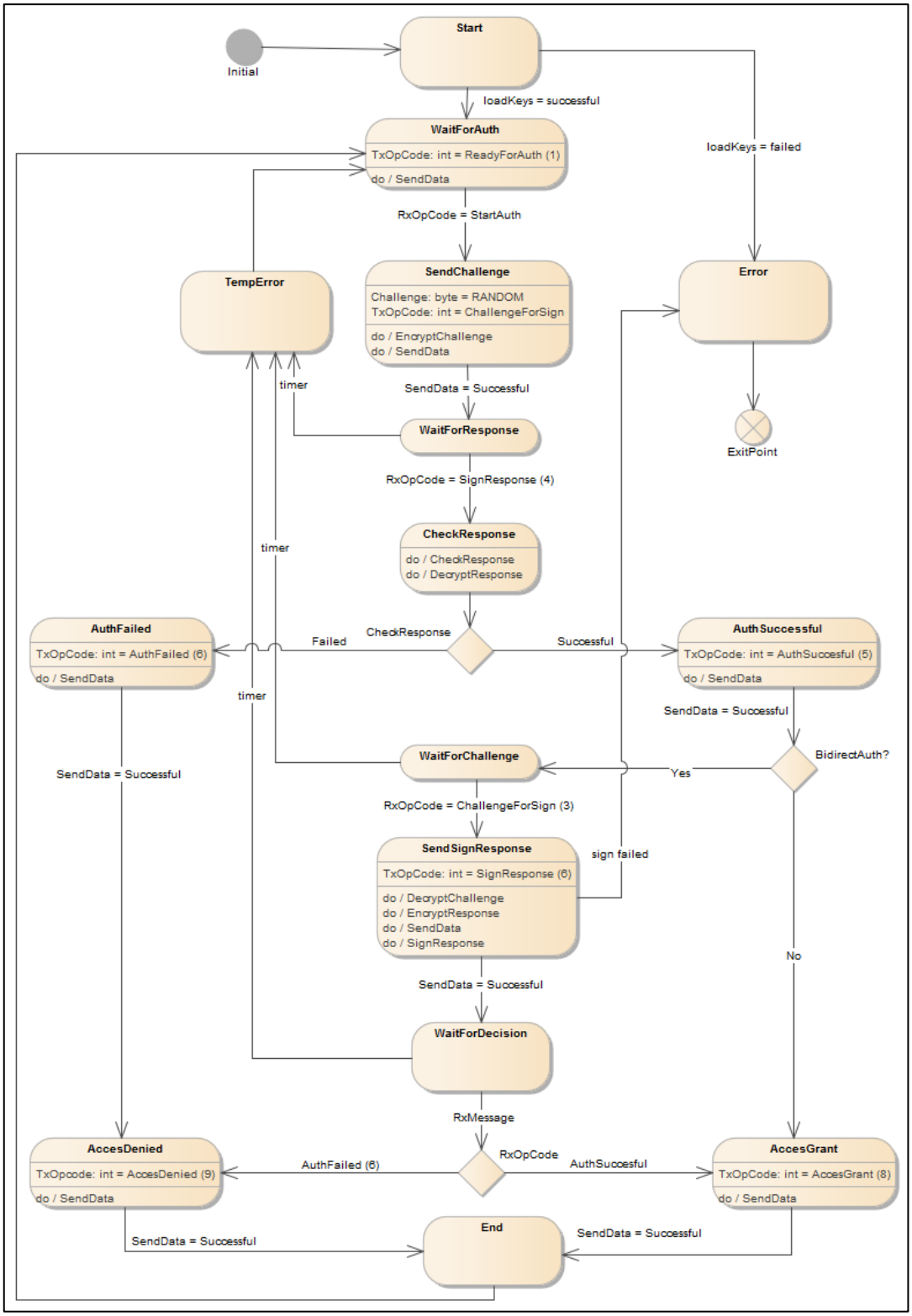

Abbildung 5.12 - Zustandsdiagramm RSU [24]

6 Entwicklung

In diesem Kapitel wird näher auf die Entwicklung, also die technische Umsetzung des in Kapitel 5 erläuterten Konzeptes eingegangen. Die Entwicklung der einzelnen Hardwarekomponenten, sowie die der Software werden dargestellt. Die Hardware ist aufgeteilt auf das Fahrzeug und den Zugangspunkt. Es wird jeweils der mechanische und elektrische Aufbau beschrieben. Zudem wird die GPIO-Ansteuerung erklärt.
Im Kapitel Software wird zunächst die Programmstruktur und die Auth-Nachricht erläutert. Es werden Codeausschnitte der wichtigsten Funktionen gezeigt und erläutert.
Zudem wird auf die Schlüsselverwaltung und die in dieser Arbeit genutzte kryptographische Bibliothek kurz eingegangen.

6.1 Fahrzeug

Da kein richtiges Fahrzeug zur Verfügung steht, wurde ein Modell als Fahrzeugersatz erzeugt. In diesem sind die Fahrzeugkomponenten integriert. Das Modell wurde so gewählt, dass es kompakt ist und einfach transportiert werden kann.

Um den optischen Eindruck eines Fahrzeuges zu bekommen, hat die Box vier angedeutete Räder in Form von Holzkreisen erhalten. Zudem wurde sie mit einer schwarzen Farbe lackiert und mit NXP Aufklebern bestückt.

6.1.1 Mechanischer Aufbau - Fahrzeug

Im inneren der Box ist ein doppelter Boden eingebaut. Unter diesem befindet sich die gesamte Verkabelung. Durch den doppelten Boden wirkt das Fahrzeug von innen aufgeräumt und kompakt.

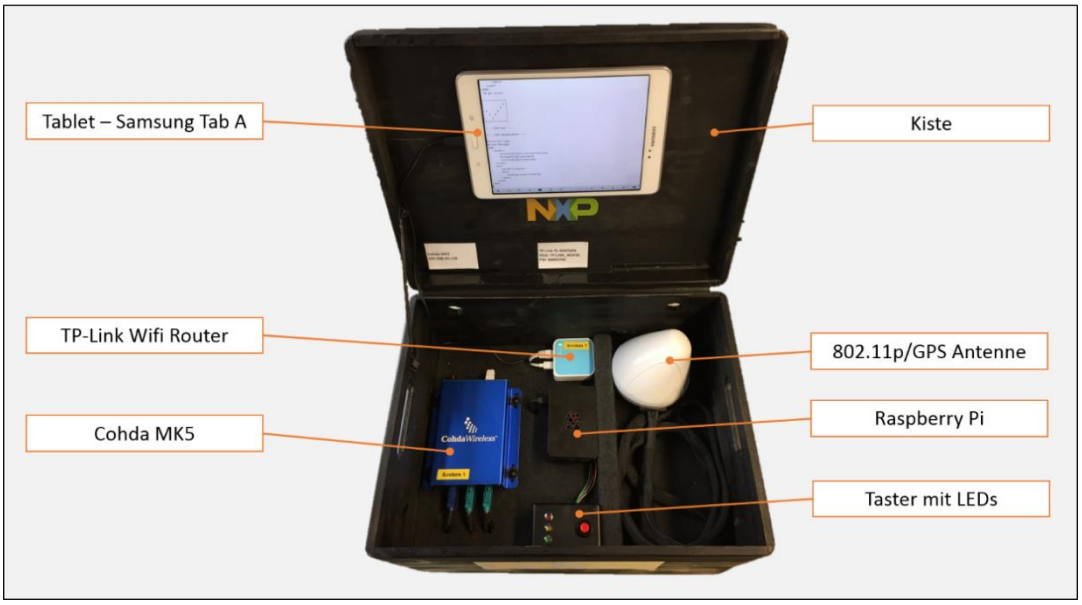

Abbildung 6.1 - Aufbau Fahrzeug [25]

In Abbildung 6.1 ist die Innenansicht des Fahrzeuges zu sehen. Man erkennt auf der linken Seite die Cohda MK5 OBU. Rechts daneben befindet sich ein Raspberry Pi 3. Dieser wird als Ethernet-to-GPIO Schnittstelle genutzt. Über dem Raspberry Pi 3 sieht man den Wifi-Router von TP-Link. Die Ethernet-to-GPIO Schnittstelle war im Konzept nicht vorgesehen. Im Laufe der Arbeit stellt sich heraus, dass es für den Demonstrator sinnvoller ist die Authentifizierung vom Fahrzeug aus zu starten.

Auf der rechten Seite ist die 802.11p/GNSS-Antenne verbaut. Diese ist herausnehmbar, sodass diese für besseren GNSS-Empfang optimal positioniert werden kann.

Im Deckel des Fahrzeugs befindet sich ein Samsung Galaxy Tab A. Auf diesem werden für die Demonstration Zusatzinformationen ausgegeben. Durch diese können alle Funktionen der Software nachvollzogen werden.

6.1.2 Elektrischer Aufbau - Fahrzeug

In Abbildung 6.2 ist die Verkabelung des Fahrzeuges dargestellt.

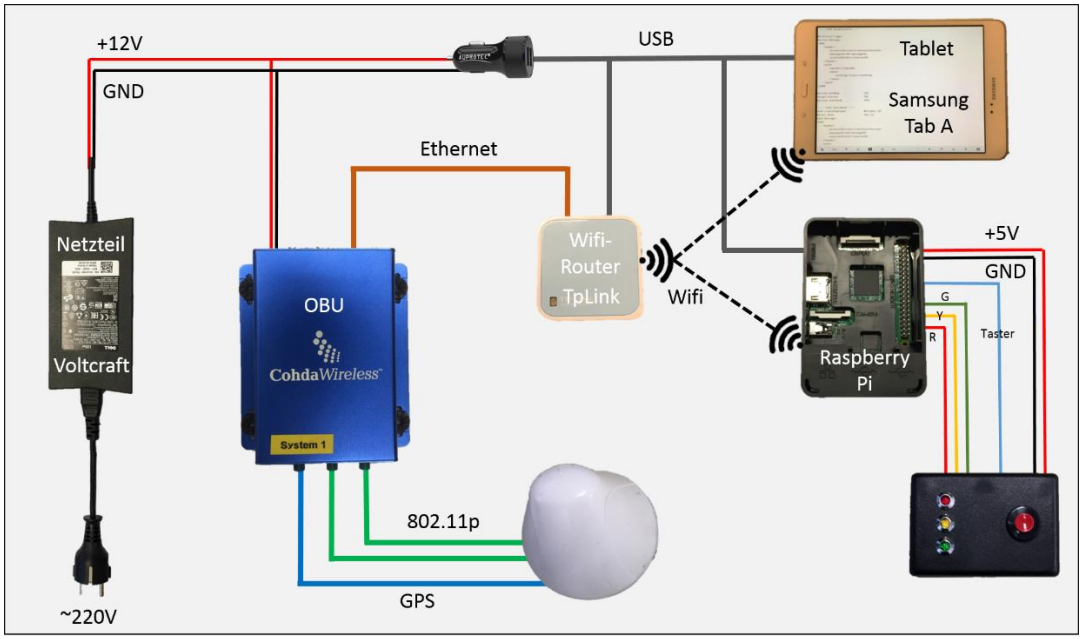

Abbildung 6.2 - Verkabelung Fahrzeug [25]

Das Netzteil wandelt die 220V Netzspannung in +12V Gleichspannung um. Die Cohda MK5 wird direkt an der +12V Spannung betrieben. An den Fakra[14]-Antennenanschluss wird die von Cohda mitgelieferte Antenne angeschlossen. In dieser sind zwei 802.11p und eine GNSS-Antenne intergiert.

Das Tablet, der Raspberry Pi und der Wifi-Router werden mit +5V als Spannung betrieben. Dafür wird aus dem Kfz Bereich ein umgebauter USB-Adapter für Zigarettenanzünder integriert. Dieser kann bis zu +24V Eingangsspannung in +5V umwandeln und liefert einen Strom von bis zu 3,4 A. An diesem werden die Komponenten über den USB-Anschluss angeschlossen.

Der Wifi-Router stellt ein Netzwerk zur Verfügung. Die Cohda MK5 wird über ein Ethernet Kabel an dieses Netzwerk angebunden. Das Tablet und der Raspberry Pi werde über Wifi an den Router angebunden.

An dem Raspberry Pi sind drei LEDs in den Farben, Rot, Gelb und Grün, sowie ein Taster angeschlossen. Diese sind in ein Kunststoffgehäuse eingearbeitet.

6.1.3 Ein- und Ausgabegerät

Als Ein- und Ausgabegerät wird ein Samsung Galaxy Tab A genutzt. Als Betriebssystem ist auf diesem Android vorinstalliert. Für die Ein- und Ausgaben auf dem Tablet wird die App „Termius[15]“ genutzt. Diese bietet die Möglichkeit eine SSH-Verbindung zur Cohda MK5 herzustellen.

[14] Steckverbinder von Fachkreis Automobil für Antennensignale im Automobilbereich [30].
[15] https://www.termius.com/

6.2 Zugangspunkt

Als Zugangspunkt wird der optische Signalgeber von Swarco genutzt. Es wurde ein Halter mit Standfuß gebaut, an dem der optische Signalgeber befestigt wird. Die gesamte Verkabelung wurde innerhalb des Signalgebers verbaut. Auf der Rückseite finden sich die Cohda MK5, sowie die GNSS/802.11p Antenne.

Als Zusatz ist das RFID-Lesegerät M-ARU von Kathrein an dem Zugangspunkt montiert. Über den Raspberry Pi, welcher für die Signalansteuerung genutzt wird, wird auch das RFID-Lesegerät gesteuert. So kann dieser Zugangspunkt auch als RFID-Demonstrator genutzt werden.

6.2.1 Mechanischer Aufbau - Zugangspunkt

In Abbildung 6.3 ist der Zugangspunkt in der Vorder- und Seitenansicht dargestellt.

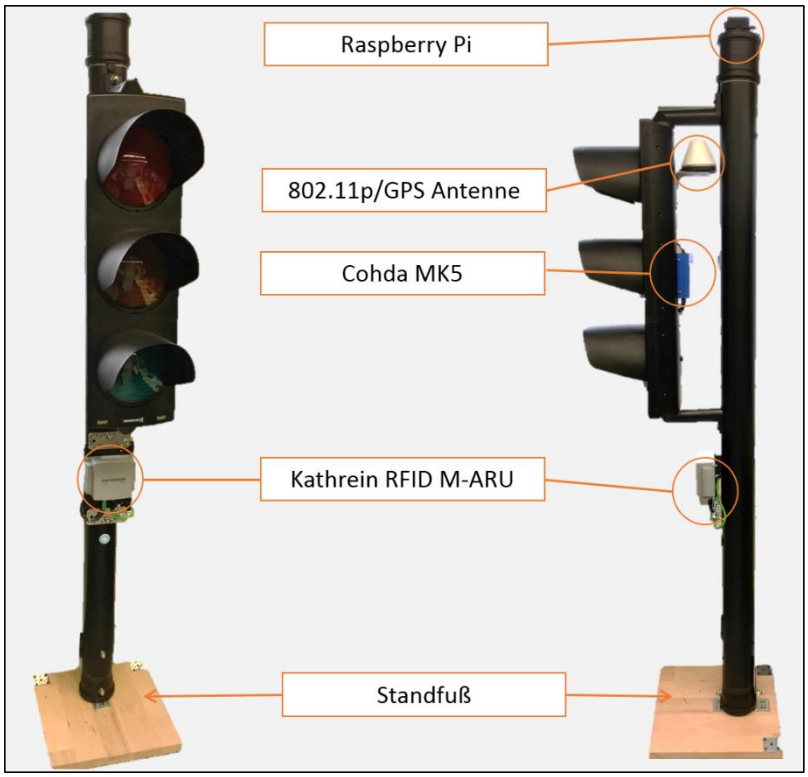

Abbildung 6.3 - Aufbau Zugangspunkt [25]

Der Aufbau steht auf einer rechteckigen Holzplatte. An dieser befinden sich für den Transport zwei drehbare Rollen. Auf der Holzplatte ist mit Metallwinkeln ein Regenrohr aus Kunststoff befestigt. Innenliegend befindet sich zur Verstärkung der Konstruktion auf ganzer Länge ein Holzkern.

Von Vorne ist lediglich der Signalgeber, sowie das RFID-Lesegerät zu sehen. Auf der Seitenansicht erkennt man, dass die Cohda MK5, sowie die dazugehörige GNSS/802.11p Antenne auf der Rückseite des optischen Signalgebers moniert sind.

Am oberen Ende der Säule befindet sich ein abnehmbarer Deckel. Unter diesem befindet sich der Raspberry Pi mit Relais-Board. Am Fuß der Säule ist ein Kabel für den Anschluss an eine 220V Steckdose herausgeführt.

6.2.2 Elektrischer Aufbau - Zugangspunkt

In Abbildung 6.4 ist die Verkabelung des Zugangspunktes dargestellt.

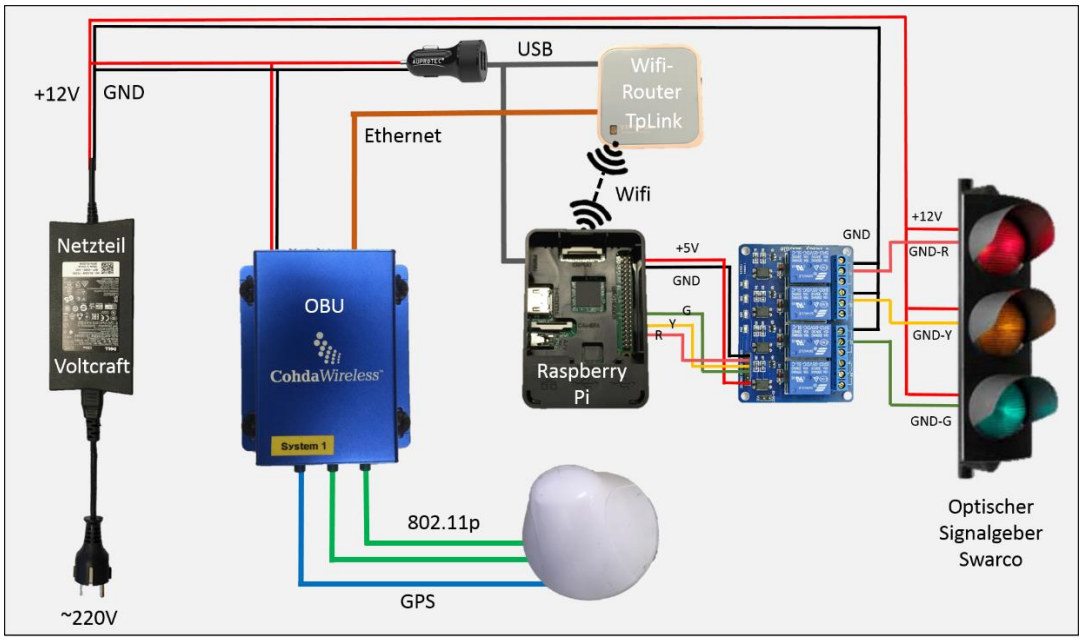

Abbildung 6.4 - Verkabelung Zugangspunkt [25]

Auch hier wird das selbe Netzteil von Voltcraft für die +12V Versorgung genutzt. An dieser ist die Cohda MK5 direkt angeschlossen. An den Fakra-Antennenanschluss wird die von Cohda mitgelieferte Antenne angeschlossen. In dieser sind zwei 802.11p und eine GNSS-Antenne intergiert.

Über den USB-Adapter werden, wie bei dem Fahrzeugaufbau beschrieben, der Wifi-Router und der Raspberry Pi versorgt. Auch hier baut der Wifi-Router ein Netzwerk auf, über welches alle Komponenten mit einander verbunden sind.

Anders als beim Fahrzeugaufbau war hier die Signalansteuerung herausfordernder (Kapitel 6.4). Infrastrukturseitig wird ein optischer Signalgeber, also das Herzstück einer Ampel von Swarco genutzt. Dessen Leuchtmittel benötigen +12V und einen höheren Strom als die einzelnen LEDs beim Fahrzeug. Daher können diese nicht direkt von dem GPIO-Ausgängen des Raspberry Pis angesteuert werden.

Für die Ansteuerung wird ein 4-Kanal Relais-Board genutzt. Dieses steuert galvanisch getrennt über Optokoppler[16] Relais an, welche wiederum die Leuchtmittel des Signalgebers ansteuern.

Die Leuchtmittel des Signalgebers werden mit +12V Spannung versorgt. Die Masseleitung wird jeweils über das Relais-Board auf Masse geschaltet.

[16] Bauelement zur Übertragung eines Signals zwischen zwei galvanisch getrennten Stromkreisen [36].

6.3 Software

Der Zustandsautomat wurde mit Hilfe der Cohda SDK programmiert, kompiliert und auf die MK5 geladen. Für den Zustandsautomaten sind zwei Funktionen implementiert. Die eine Funktion ist „evalStates()“ und wertet die Zustände aus. Die andere Funktion heißt „evalEvents()“ und ist für die Auswertung der Übergänge zuständig. Die folgenden Codeausschnitte sind aus der originalen Software.

6.3.1 Programmstruktur

Die Software wird AuthApp genannt, da sie für die Authentifizierung genutzt wird. Die bestehende Programmstruktur der exampleETSI wird genutzt und erweitert. Für die AuthApp ergibt sich folgenden Ordnerstruktur. Die Ordnerstruktur wird wegen der Übersicht vereinfacht dargestellt.

- **AuthApp**
  - AuthApp (Ausführbares Programm)
  - AuthApp-mk5.tgz (Packet zum Übertragen auf die MK5)
  - Makefile (Makefile zum kompilieren)
  - raw.cfg (Konfigurationsdatei)
  - raw.conf (Konfigurationsdatei)
  - rc.AuthApp (Skript zum Starten der Software)
  - rsa_priv_OBU.txt (Privater Schlüssel der OBU)
  - rsa_pub_OBU.txt (Öffentlicher Schlüssel der OBU)
  - rsa_priv_RSU.txt (Privater Schlüssel der RSU)
  - rsa_pub_RSU.txt (Öffentlicher Schlüssel der OBU)

  - **v2x-lib** **(Ordner für V2X-Bibliotheken)**
    - **include** **(Ordner mit includes)**
    - **lib** **(Ordner mit Bibliotheken)**
      - **mk5** **(Ordner mit Bibliotheken für die MK5)**

  - **Src** **(Ordner mit den Code Dateien)**
    - main.c (Quellcode für die Main)
    - RAW.asn1 (Definition der AUTH-Nachricht in ASN1)
    - raw-its.c (Quellcode der Auth-Software)
    - raw-its.h (Header-Datei zur raw-its.c)
    - rsa.c (Quellcode für die RSA-Signierung)
    - rsa.h (Header-Datei zur rsa.c)
    - **mbedtls** **(Ordner mit Bibliothek für Kryptografie)**
    - **gen** **(Generierte Quellcode aus ASN.1)**

Ordner: **Fett markiert**
Hinzugefügte Dateien: Grau markiert
veränderte Dateien: Rot markiert

6.3.2 Auth-Nachricht

Das in Kapitel 5.1.2 beschriebene Konzept der Auth-Nachricht wird in ASN.1 umgesetzt.
Die Definition wurde in der RAW.asn1 Datei geschrieben, da diese bereits in dem Programm eingebunden ist.
Im Rahmen dieser Arbeit reicht es die vorhandene Datei zu verändern. Aufgrund der eingebundenen Datei in Programmstruktur, konnte der Name „RAW“ nicht in „AUTH“ umbenannt werden. Daher findet sich im weiteren Verlauf der Bezeichner „RAW“ in den Codeausschnitten.
Im Anschluss an diese Arbeit kann die Definition der Auth-Nachricht in eine eigene Datei ausgelagert werden. Im folgenden Codeausschnitt ist die Definition der Auth-Nachricht in ASN.1 dargestellt.

In Code 6.1 – Z. 8ff. ist die Unterteilung der Auth-Nachricht in den ItsPduHeader und in die AuthMsg gezeigt. Der ItsPduHeader besteht wiederum aus der protocolVersion, der messageID und stationID.

```
1  RAW-PDU-Descriptions {
2  version (1)
3  }
4
5   DEFINITIONS AUTOMATIC TAGS ::=
6   BEGIN
7
8   RAW ::= SEQUENCE {
9       header  ItsPduHeader,
10      auth    AuthMsg
11  }
12
13  ItsPduHeader ::= SEQUENCE {
14      protocolVersion INTEGER{ currentVersion(1) } (0..255),
15      messageID       INTEGER{ denm(1), cam(2), poi(3), spat(4), map(5)
16                               ivi(6), ivs(7), ev-rsr(8), dummy(99),
17                               auth(100)} (0..255),
18
19      stationID       StationID
20      }
21
22      StationID ::=   INTEGER(0..65535)
23
24      AuthMsg ::= SEQUENCE {
25              opcode  INTEGER{ noopcode(0), readyforauth(1), startauth(2),
26                               startbiauth(3), challengeforsign(4),
27                               signresponse(5), authsuccessful(6),
28                               authfailed(7), accessgrant(8),
29                               accessdenied(9) } (0..255),
30              data Data,
31              ...
32      }
33
34      Data ::= CHOICE {
35                      challenge      Challenge,
36                      response       Response,
37                      nothing        Nothing,
38                      cryptoAlgoType INTEGER{ nocrypto (0), aes128(1),
39                                              aes256(2) } (0..255),
40                      ...
41      }
42
43      Challenge ::= OCTET STRING (SIZE(1..8))
44      Response ::= OCTET STRING (SIZE(1..266))
45      Nothing ::= BOOLEAN
46
47      END
```

Code 6.1 - Auth.asn1 [Anhang A: RAW.asn1]

ItsPduHeader [Code 6.1 – Z. 13]

protocolVersion
Integerwert zwischen 0 und 255. Die (1) steht für currentVersion.

messageID
Integerwert zwischen 0 und 255. Die (1) steht für ein DENM, die (2) für eine CAM. Im Rahmen dieser Arbeit wurde für die Auth-Nachricht die MessageID 100 definiert [Code 6.1 – Z. 18].

StationID
Integerwert zwischen 0 und 65535.

AuthMsg [Code 6.1 – Z. 24]

Opcode
Integerwert zwischen 0 und 255. Die Nummern der Opcodes sind im Kapitel 5.1.3 definiert und sind genauso in der ASN.1 Definition eingebracht.

Data
Ist eine Auswahl zwischen Challenge, Response, Nothing und cryptoAlgoType.

Data [Code 6.1 – Z. 34]

Challenge
Octet String ist zwischen 1 und 8 Zeichen lang. Die Challenge ist eine Zufallszahl die 8 Bytes groß ist. Daher reicht eine Octet String der maximalen Größe von 8.

Response
Octet String ist zwischen 1 und 266 Zeichen lang. Als Response wird der signierte Hashwert der Challenge mit den hinzugefügten Zufallszahlen geschickt. Dieser hat eine maximale Größe von 266 Byte.

Nothing
Ist ein Booleanwert und kann 0 oder 1 sein. Dieser dient als Standardwert, wenn keine Daten verschickt werden.

CryptoAlgoType
Integerwert zwischen 0 und 255. Gibt den Verschlüsselungstyp an.
(0) bedeutet keine Verschlüsselung, (1) steht für AES128 und (2) für eine AES256 Verschlüsselung. Dies sind nur Beispiele an Verschlüsselungsalgorithmen und können beliebig erweitert werden.

In Abbildung 6.5 wird eine vereinfachte Version des Zustandsdiagramms gezeigt. In diesem sind die Nachrichten eingezeichnet, welche für die Authentifizierung ausgetauscht werden. Die Nachrichten wurden mit der Software Wireshark aufgezeichnet.

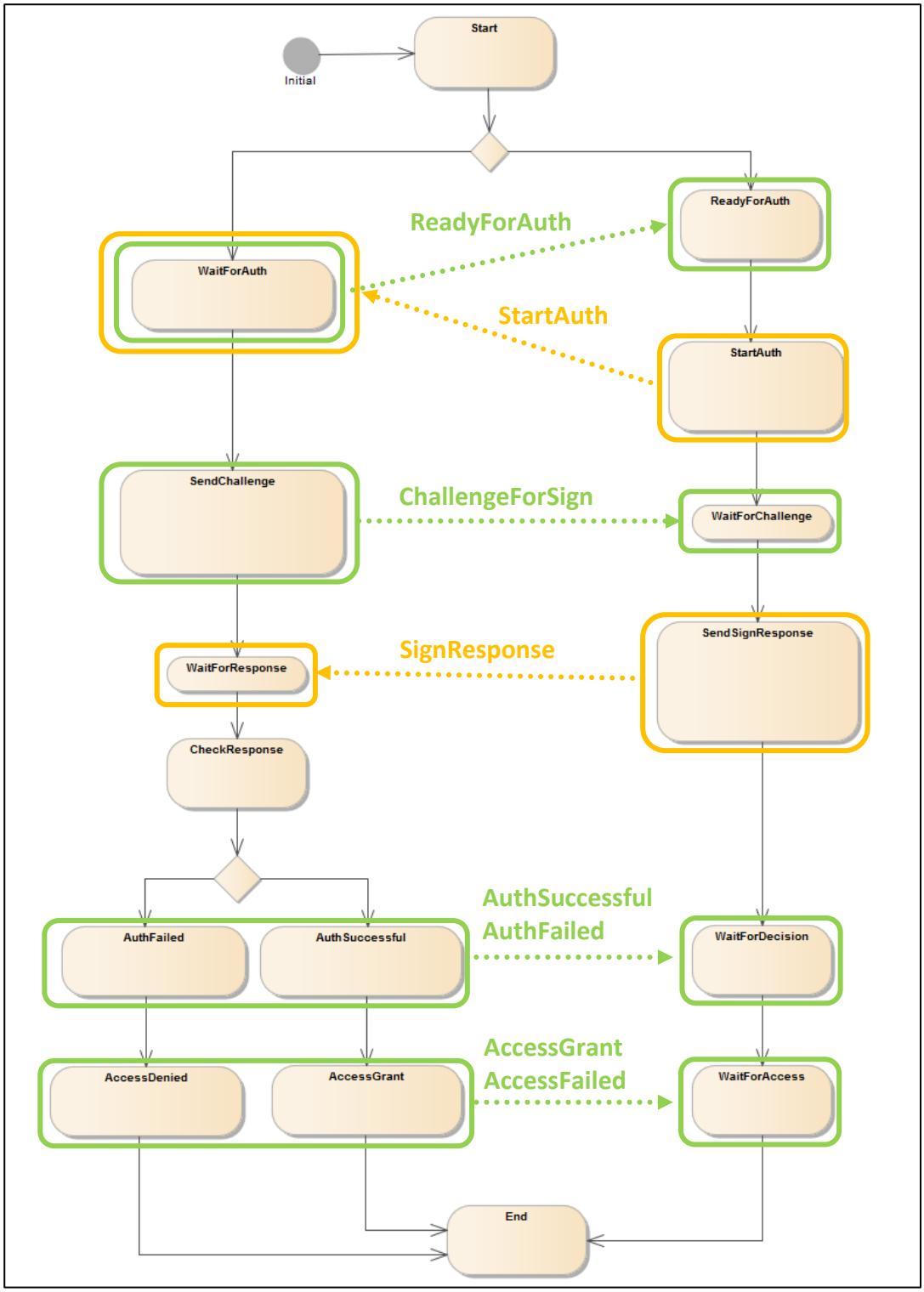

Abbildung 6.5 - Nachrichtenaustausch [24]

Der Header der Nachrichten besteht wie schon beschrieben aus der protocolVersion, der messageID und der stationID. Bis auf die stationID ist der Header bei allen Nachrichten identisch.
Die RSU besitzt als stationID die „1011“ und die OBU die „5115“.

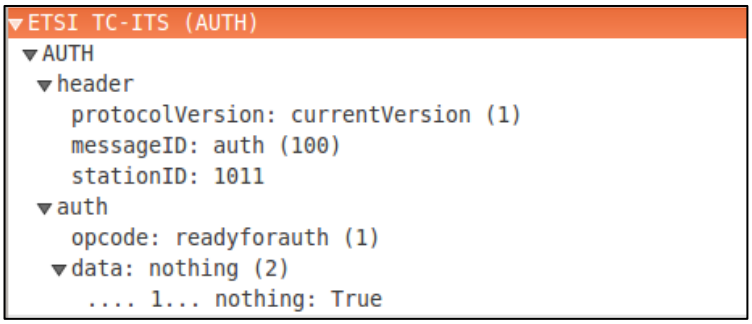

Abbildung 6.6 - Log Datei ReadyForAuth [Anhang A: Wireshark_Logs]

Die Nachricht in Abbildung 6.6 mit dem Opcode „ReadyForAuth“ wird von der RSU zyklisch versendet.

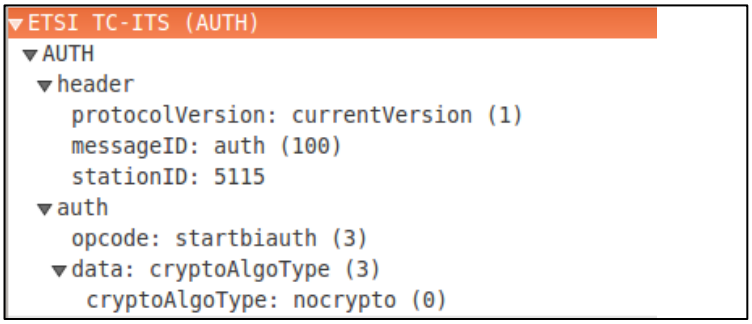

Abbildung 6.7 - Log Datei StartAuth [Anhang A: Wireshark_Logs]

Von der OBU wird eine Nachricht mit dem Opcode „StartAuth“ zum Starten einer Authentifizierung geschickt. Als „data“ wir dieser Nachricht der Verschlüsselungstyp angehängt.

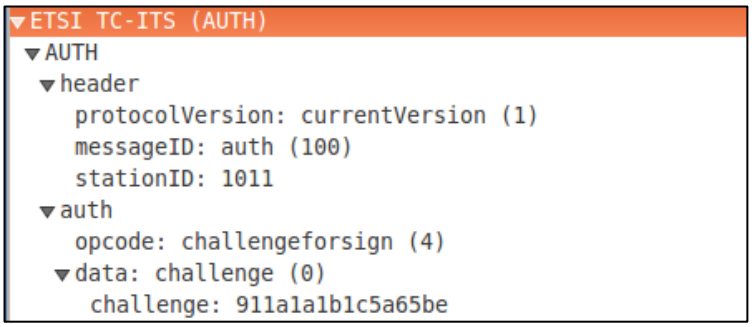

Abbildung 6.8 - Log Datei ChallengeForSign [Anhang A: Wireshark_Logs]

Die RSU generiert daraufhin eine Zufallszahl und schickt diese in „data“ als Challenge [Abbildung 6.8].

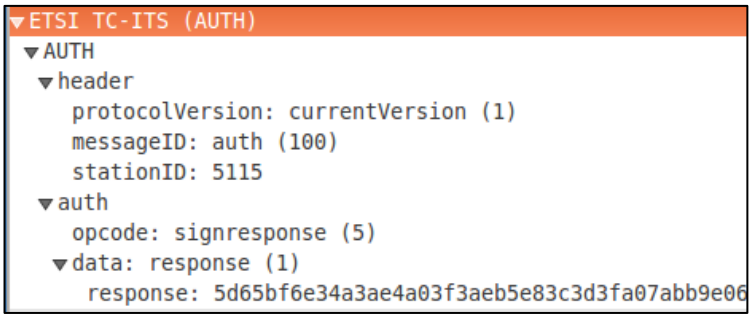

Abbildung 6.9 - Log Datei SignResponse [Anhang A: Wireshark_Logs]

Die Challenge wird von der OBU signiert und als Response in „data“ an die RSU gesendet [Abbildung 6.9].

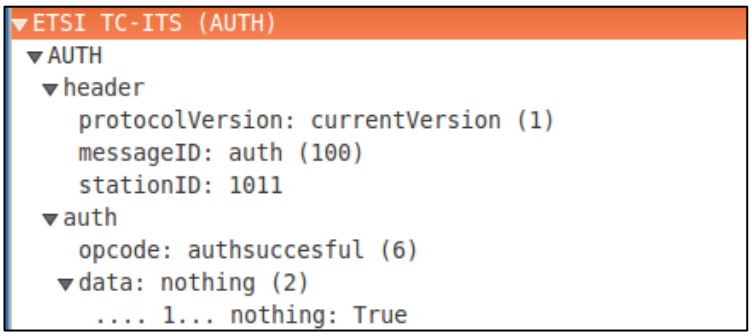

Abbildung 6.10 - Log Datei AuthSuccessful [Anhang A: Wireshark_Logs]

Wenn die Signierung von der RSU verifiziert wurde, sendet die RSU eine Bestätigung mit dem Opcode „AuthSuccessful“ an die OBU [Abbildung 6.10].
Zum Schluss sendet die RSU eine Nachricht an die OBU, um den Zugang freizugeben.

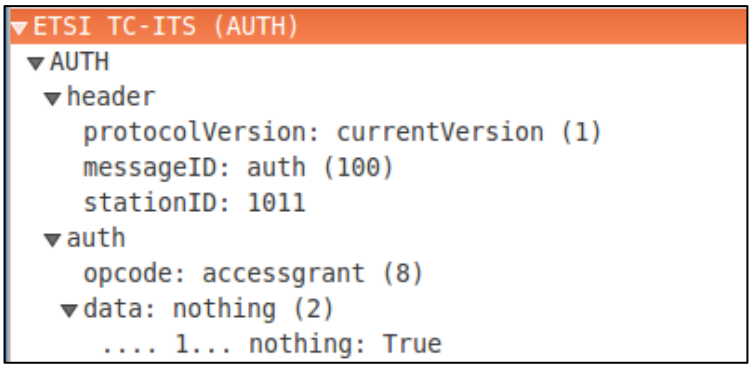

Abbildung 6.11 - Log Datei AccessGrant [Anhang A: Wireshark_Logs]

Dazu wird als Opcode „AccessGrant“ genutzt [Abbildung 6.11].

6.3.3 Thread-Loop

In Code 6.2 ist die while-Schleife angezeigt, welche zyklisch aufgerufen wird. Dort wird zuerst die Funktion evalStates() aufgerufen. In dieser werden die Zustandsaktionen ausgeführt.
Danach wird überprüft, ob das Flag „flagTxAuth“ gesetzt wurde. Sollte dies der Fall sein, wird die Funktion RAWIts_ReqSend() aufgerufen.
Diese versendet eine Auth-Nachricht. Im Anschluss wird die Funktion UtilNap() aufgerufen. Diese sorgt für eine Verzögerung, bis die Nachricht versendet wurde.

Zum Schluss wird die Funktion evalEvents() aufgerufen. Dort wird überprüft, ob Übergangsbedingungen zum Wechseln des Zustandes erfüllt sind.

```
1700  while ((pRAW->ThreadState & RAWITS_THREAD_STATE_STOP) == 0)
1701  {
1702
1703   evalStates();
1704
1705   if(flagTXAuth == 1){
1706          RAWIts_ReqSend(RAWHandle, &pRAW->Params);
1707          UtilNap(pRAW->Params.TxInterval, &Time);
1708   }
1709
1710   evalEvents();
1711
1712  }
```

Code 6.2 - Thread Loop [Anhang A: raw-its.c]

6.3.4 RAWIts_ExtCallback()

Die Funktion RAWIts_ExtCallback() wird aufgerufen, wenn eine neue Nachricht empfangen wurde. Diese wird in der Funktion decodiert und in einer Struktur abgespeichert.

```
1877    // Decode (from PER), decoded buffer will be created
1878    void *pDec = NULL;
1879    asn_dec_rval_t rval = uper_decode_complete(0,
1880                                               &asn_DEF_RAW,
1881                                               &pDec,
1882                                               pPayload,
1883                                               Length);
1884    if (rval.code == RC_OK)
1885    {
1886      fprintf(stderr, "RAW Decode OK\n");
1887      xer_fprint(stderr, &asn_DEF_RAW, pDec);
1888
1889      pRawRX = (RAW_t *)pDec;
1890
1891     if(((int)pRawRX->header.messageID==MESSAGE_ID_AUTH)&&(flagRXAuth== 0)){
1892          flagRXAuth = 1;
1893          printf("Recieve Message: \n");
1894          xer_fprint(stdout, &asn_DEF_RAW, pDec);
1895     }
```

Code 6.3 - RAWIts_ExtCallback [Anhang A: raw-its.c]

Im Code 6.3– Z. 1879 wird die empfangene Nachricht decodiert und in Struktur „pDec“ gespeichert. Die Struktur „pRawRX“ ist eine Struktur vom Typ RAW_t, welche global initialisiert wurde. Dadurch kann aus den Zuständen direkt auf diese zugegriffen werden.
Im Code 6.3 – Z. 1889 wird „pDec“ auf „pRawRX“ kopiert.
Es wird überprüft, ob die empfangene Nachricht ein Auth-Nachricht ist und falls dies der Fall ist, wird das Flag „flagRXAuth“ auf „1“ gesetzt. Dadurch weiß der Zustandsautomat, dass einen neue Auth-Nachricht empfangen wurde.

6.3.5 RAWIts_ReqSend()

In der Funktion RAWIts_ReqSend() wird der BTP-Header mit den nötigen Informationen gefüllt. Darauf wird in dieser Arbeit nicht näher eingegangen.

Von RAWIts_ReqSend() wird die Funktion RAW_MsgCreate() [Code 6.4 – Z.1789] aufgerufen. Dort wird die eigentliche Nachricht mit Informationen gefüllt.

Wenn dies erfolgreich war, wird die Nachricht für Ausgabe decodiert [Code 6.4 – Z.1798] und für das Versenden mit dem ASN.1 PER Encoder encodiert [Code 6.4 – Z. 1803.

Zum Schluss wird das Nachrichtenpaket mit der Funktion ITS_SendMSGPacket() gesendet [Code 6.4 – Z. 1827].

```
1788  // Create and populate the message
1789  int Res = RAW_MsgCreate(&pMsg, Params->DataLength, Params->Data);
                    .
                    .
                    .
1797  // Use the XER decoder to show the message in the debug output
1798   xer_fprint(stderr, &asn_DEF_RAW, pMsg);
1799   printf("Send Message: \n");
1800   xer_fprint(stdout, &asn_DEF_RAW, pMsg);
1801
1802  // Encode (with PER), we need to provide a buffer for the encoder to use
1803  asn_enc_rval_t EncRVal = uper_encode_to_buffer(&asn_DEF_RAW,
1804                                                 pMsg,
1805                                                 pBuf,
1806                                                 RAW_ENC_BUFFER_SIZE);
                    .
                    .
                    .
1826  // And send
1827  Stat = ITS_SendMSGPacket(&MSGReqHdr, pBuf, (EncRVal.encoded+7)/8 );
```

Code 6.4 - RAWIts_ReqSend [Anhang A: raw-its.c]

6.3.6 RAW_MsgCreate()

In der Funktion RAW_MsgCreate() wird die in ASN.1 definierte Auth-Nachricht mit Informationen gefüllt.

```
1928 static int RAW_MsgCreate(RAW_t **ppMsg, int Len, uint8_t *Data) {
1929
1930  //static uint16_t Seq = 0;
1931  int RetVal = -ENOSYS;
1932
1933  // Allocate memory for the main RAW t
1934  *ppMsg = (RAW_t *)calloc(sizeof(RAW_t),1);
1935  if (!*ppMsg)
1936  {
1937    RetVal = -ENOMEM;
1938    goto Error;
1939  }
1940
1941  // set the PDU header
1942  (*ppMsg)->header.protocolVersion
              =ItsPduHeader_protocolVersion_currentVersion;
1943  (*ppMsg)->header.messageID = VAR_MessageID;
1944  (*ppMsg)->header.stationID = VAR_StationID;
1945
1946  (*ppMsg)->auth.opcode = VAR_opcode;
1947  (*ppMsg)->auth.data.present = VAR_DataPresent;
```

Code 6.5 - RAW_MsgCreate 1 [Anhang A: raw-its.c]

Zunächst wird für die Struktur „RAW_t“ Speicherplatz reserviert [Code 6.5– Z.1934]. Danach wird der Header der Auth-Nachricht gefüllt. In der Software sind Variablen initialisiert, welche in den Zuständen mit Informationen gefüllt werden. Diese Variablen werden in der RAW_MsgCreate() in die Struktur kopiert. So muss der Zustandsautomat keinen direkten Zugriff auf die Struktur haben. Dies vereinfacht den Code erheblich.

```
1949 switch (VAR_DataPresent){
1950
1951      case Data_PR_challenge:
1952      if(((*ppMsg)->auth.data.choice.challenge.buf) == NULL) {
1953
1954            (*ppMsg)->auth.data.choice.challenge.buf =
                        calloc(sizeof(unsigned char),
                        sizeof(challenge));
1955
1956            if(!(*ppMsg)->auth.data.choice.challenge.buf)
1957            {
1958              RetVal = -ENOMEM;
1959              goto Error;
1960            }
1961            memcpy((*ppMsg)->auth.data.choice.challenge.buf,
1962                          challenge,
1963                          sizeof(challenge));
1964
1965            (*ppMsg)->auth.data.choice.challenge.size = sizeof(challenge);
1966      }
1967      break;
```

Code 6.6 - RAW_MsgCreate 2 [Anhang A: raw-its.c]

In Code 6.6 wird das „data“, welches als Auswahl definierte ist, gefüllt. Mit der Variablen VAR_DataPresent wird der Inhalt von Data ausgewählt. Mit einer „switch-case“- Anweisung wird zwischen den verschiedenen Auswahlen unterschieden. Je nachdem wie Data gewählt ist, wird diese mit unterschiedlichem Inhalt gefüllt.
Wenn das Füllen der Nachrichtenstruktur erfolgreich war, wird das FlagTXAuth auf null gesetzt. So wird dem Zustandsautomaten mitgeteilt, dass die Nachricht erfolgreich erstellt und versendet wurde.

6.3.7 #defines

```
12  //--------------------------------------------------------------------------
13  // define RSU for RSU-Version / define OBU for OBU-Version
14  // define MUTUAL for mutual Authentication
15  //--------------------------------------------------------------------------
16
17  //#define RSU
18  #define OBU
19
20  #define MUTUAL
21
```

Code 6.7 - #define OBU/RSU [Anhang A: raw-its.c]

Für die Auswahl zwischen OBU- und RSU-Version wird ein „#define“ verwendet.
Vor dem Kompilieren wird im Quellcode „OBU“ oder „RSU“ per „#define“ gesetzt [Code 6.7 – Z. 18]. Das Andere wird auskommentiert.
Zudem wird per „#define“ ausgewählt, ob eine einseitige oder gegenseitige Authentifizierung durchgeführt wird [Code 6.7 – Z. 20].

Im Quellcode Anhang A raw-its.c [Z. 63 - 117] sind alle „#defines“ für die Auth-Software aufgeführt. Diese werden anstelle von sogenannten Magic Numbers verwendet. Dadurch kann die Software im Nachhinein einfach verändert und erweitert werden. Zudem verhelfen „#defines“ zur besseren Lesbarkeit und machen den Code verständlicher.

6.3.8 Variablen

Im Quellcode Anhang A raw-its.c [Z. 135 - 212] werden die global initialisierten Variablen aufgezeigt. Es wird auf die wichtigsten Variablen der Software näher eingegangen. Die Restlichen sind selbsterklärend.

Auf den Variablen „myStationID“ und „targetStationID“ wird eine Kennzahl als Identifikationsnummer gespeichert. Je nachdem ob es sich um eine OBU oder RSU handelt werden diese auf einer der beiden Variablen gespeichert.

Die Variablen „BufForResponseHash“ und „BufForChallengeHash“ sind als Zwischenspeicher für die Zufallszahlen um daraus den Hashwert zu berechnen.

„flagTXAuth“ und „flagRXAuth“ sind Variablen, welche als Merker genutzt werden, um das versenden oder ankommen von Nachrichten zu markieren.

Die Variablen „VAR_...“ sind Variablen, welche für das Befüllen der Nachrichten Struktur von der „RAW_MsgCreate()“ Funktion genutzt werden. Die Variablen „challenge“ und „response“ gehören auch zu dieser Gruppe von Variablen.

„RxChallenge“, „TxChallenge“, „RxResponse“ und „TxResponse“ sind Variablen, welche als Zwischenspeicher für das Bearbeiten innerhalb der FSM genutzt werden.

Die Variable „flagPrint“ wird als Marker für einmalige Ausgaben auf der Konsole innerhalb eines Zustandes genutzt.

6.3.9 Zustandsautomat

Die Zustände des Zustandsautomaten und deren Funktionen wurden im Kapitel 5.3 näher erläutert, daher wird in diesem Kapitel die Implementierung dieser Funktionen aufgezeigt. Zu jedem Zustand wird der dazugehörige Code aus der „evalStates()“ für die Zustandsaktionen und der dazu gehörige Code für die Zustandsübergänge aus „evalEvents()“ aufgezeigt.

Die Zustände sind in als Enumeration definiert [Code 6.8]. Sie werden in der Variable „currentState“ gespeichert. In den Funktionen „evalStates()“ und „evalEvents“ wird auf „currentState“ eine „switch-case“ Anweisung ausgeführt.

```
224 //------------------------------------------------------------------------
225 //Finale State Machine
226 //------------------------------------------------------------------------
227
228 enum states {
229     Start,
230     WaitForAuth,
231     ReadyForAuth,
232     StartAuth,
233     WaitForChallenge,
234     SendChallenge,
235     WaitForResponse,
236     SendSignResponse,
237     WaitForDecision,
238     CheckResponse,
239     AuthSuccessful,
240     AuthFailed,
241     WaitForAccess,
242     AccessGrant,
243     AccessDenied,
244     End,
245     Error,
246     TempError
247     }currentState;
```

Code 6.8 - Enumeration Zustände [Anhang A: raw-its.c]

Für den Demonstrator ist die Software so aufgebaut, dass diese auf einen Triggersignal wartet und erst dann eine Authentifizierung startet. Im Zustand ReadyForAuth ist eine do-while-Schleife implementiert, die zyklisch den Trigger abfragt. Erst bei Betätigung dieses Triggers wird eine Authentifizierung ausgelost. Dadurch können die OBU und die RSU in unmittelbarer Nähe zu einander stehen, ohne dass unmittelbar eine Authentifizierung stattfindet.

Beispielhaft werden für den Zustand WaitForChallenge die Aktionen in evalStates() und die Übergangsbedingungen in evalEvents() dargestellt und erläutert. Die restlichen Zustände sind ähnlich diesem aufgebaut. Lediglich die Aktionen und die Übergangsbedingungen unterscheiden sich.

Zustandsaktionen

```
552     case WaitForChallenge:
553
554             if(flagPrint == 0){
555                     clock_gettime(CLOCK_REALTIME, &_ts);
556                     start_time = _ts.tv_sec;
557                     printf("\n------CASE_WaitForChallenge------\n");
558                     flagPrint = 1;
559             }
560
561             clock_gettime(CLOCK_REALTIME, &_ts);
562             end_time = _ts.tv_sec;
563             if(end_time-start_time == i){
564                     printf("%i\n",i);
565                     i++;
566             }
567
568     break;
```

Code 6.9 - Zustandsaktionen [Anhang A: raw-its.c]

Beim erstmaligen Eintreten in einen Zustand, erfolgt eine Ausgabe auf der Konsole, sodass erkennbar ist in welchem Zustand sich die Software befindet [Code 6.9 – Z.557].
Zudem wird die Zeit in Sekunden abgespeichert, in der der Zustand aufgerufen wird.
Solange die Software in diesem Zustand verbleibt, wird die aktuelle Zeit erfasst. Während der Authentifizierung ist ein Timeout für die Zustände implementiert. Dadurch wird verhindert, dass die Software sich in einer Endlosschleife verfängt und auf Nachrichten wartet.
Nach der Zeiterfassung können weitere Aktionen, wie das Vorbereiten von Auth-Nachrichten erfolgen.

Übergangsbedingungen

```
1121    case WaitForChallenge:
1122
1123            if(flagRXAuth == 1){
1124                    if(pRawRX->auth.opcode == OPCODE_CHALLENGEFORSIGN){
1125
1126                            printf("Receive Challenge            [X]\n");
1127                            flagPrint = 0;
1128                            currentState = SendSignResponse;
1129                            }
1130                    flagRXAuth = 0;
1131            }
1132
1133            if(end_time > (start_time + WAITTIMERESPONSE)){
1134                    flagPrint = 0;
1135                    currentState = TempError;
1136            }
1137
1138    break;
```

Code 6.10 - Übergangsbedingungen [Anhang A: raw-its.c]

In der Übergangsbedingung wird überprüft, ob eine Auth-Nachricht empfangen wurde [Code 6.10 – Z.1123]. Sollte dies der Fall sein, wird der Opcode der empfangenen Nachricht analysiert.
Wenn der Opcode dem erwartenden Opcode entspricht, werden die Informationen der Nachricht lokal abgespeichert.
Es folgt ein Zustandswechsel in den nächsten Zustand.
Vorher wird überprüft, ob die Zeitobergrenze überschritten wurde. Falls dies der Fall ist, wird in den Zustand TempError gewechselt.
Dort wird die Software neu gestartet.

6.4 GPIO-Ansteuerung

Für die Ansteuerung des optischen Signalgebers, sowie für das Triggersignal werden GPIO-Pins benötigt, also einzelne digitale Steuerleitungen. Sie werden z.B. zum Ein- und Ausschalten von LED Signalleuchten oder zum Abfragen eines Tasters gebraucht. Bei der Cohda MK5 sind die GPIO-Pins nicht nach außen geführt.

Die MK5 müsste aufgeschraubt werden und auf der Platine müssten die GPIO-Pins des Mikrokontrollers gesucht werden. An diesen müsste ein Abgriff aufgelötet werden. Zudem müsste eine Schutzschaltung entwickelt werden, um den Mikrokontroller zu schützen, sodass das Nutzen dieser GPIOs mit einem erheblichen Aufwand und Risiko verbunden wäre.

Daher wird für die GPIO-Ansteuerung ein Raspberry Pi 3 genutzt. Dieser wird über eine Ethernet TCP/IP Verbindung mit der Cohda MK5 verbunden. Es wurde daher im Rahmen dieser Arbeit noch zusätzlich jeweils eine Server-/ und eine Client-Software implementiert.

Die Client-Software ist mit in der Authentifizierungssoftware implementiert. Während der Initialisierung im Zustand Start, wird eine Verbindung zum Server auf dem Raspberry Pi aufgebaut. Erst bei erfolgreich hergestellter Verbindung erlaubt die Software eine Authentifizierung.
Die Server-Software wurde auf dem Raspberry Pi implementiert und kompiliert. Die Software wurde „GpioMK5“ genannt.

Sobald eine Verbindung hergestellt wurde, sendet die MK5 eine Zahl. Diese Zahl wird auf dem Server als „Waittime“ gespeichert und bestimmt wie lange die GPIOs geschaltet werden. So muss nur ein Code zum Einschalten geschickt werden. Das Ausschalten übernimmt der Server. Das Empfangen eines Codes wird vom Server bestätigt. Zudem wird beim Ausschalten der GPIOs eine Information an die MK5 gesendet.

Zum signalisieren, dass die Software bereit für eine Authentifizierung ist, werden alle GPIOs nacheinander kurz angesteuert.

6.4.1 Steuercodes

Tabelle 6.1 - GPIO Codes

FARBE	CODE
ROT	100
ORANGE	010
GRÜN	001
ALLE AUS	000
TRIGGERSIGNAL	1000

Im Quellcode Anhang A raw-its.c Z. 280 – 310 ist der Verbindungsaufbau und die Ansteuerung der GPIOs implementier.

Zum Ansteuern der GPIOs wurden Konstanten definiert. Diese sind in Tabelle 6.1 beschrieben. Der Code wurde so ausgewählt, dass die erste Zahl den Zustand von der obersten Lampe definiert. Die zweite Zahl definiert die Lampe darunter und die dritte Zahl definiert die unterste Lampe. Ein Code mit vier Stellen wir als Abfrage von GPIOs definiert.

Im Quellcode Anhang A Main_GPIOMK5.c ist die Implementierung der Serversoftware gezeigt. Diese wertet den empfangenen Code in einer switch-case-Anweisung aus und schaltet die GPIOs.

6.4.2 Elektrischer Aufbau

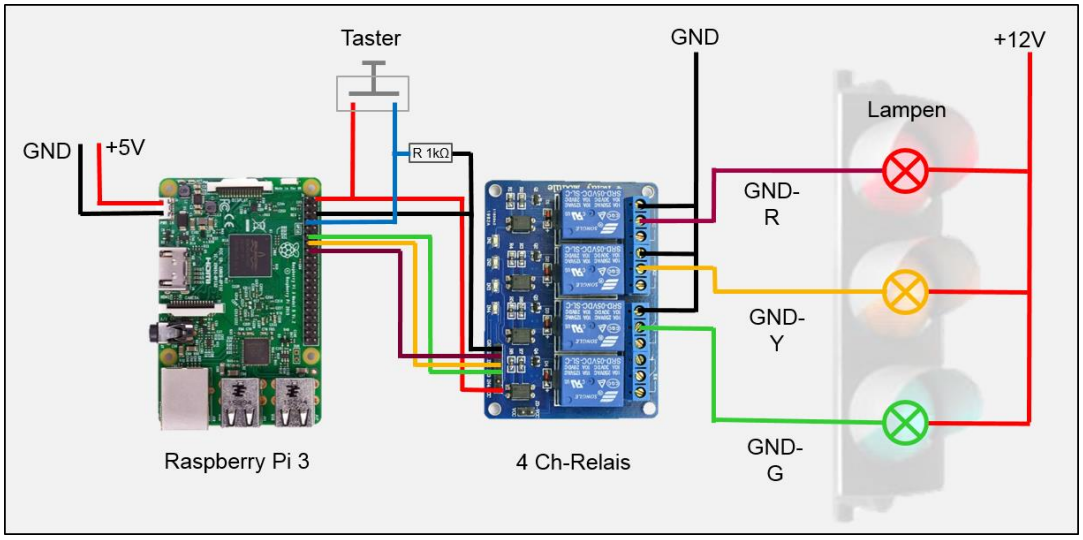

Abbildung 6.12 - Elektrischer Aufbau Signalansteuerung [25]

In Abbildung 6.12 ist der Schaltplan der Verkabelung des Raspberry Pi mit dem Relay Board und dem optischen Signalgeber dargestellt. In dem Schaltplan sind LEDs anstelle der Leuchtmittel des Signalgebers dargestellt.

Der Raspberry Pi wird über den Micro-USB-Anschluss mit +5V Spannung versorgt. An Pin 2 der Stiftleiste des Raspberry Pi liegen +5V an. Dieser Pin wird für die Versorgung des Relais-Boards und für den Taster genutzt.

Die Masse befindet sich auf Pin 6. Dort wird die Masse des Relais-Boards angeschlossen. Zudem wird auf diesen ein Pull-Down Widerstand auf dem Tastereingangs Pin geschaltet. Dadurch wird der Pegel am Eingang bei nicht bestätigtem Taster auf Massepotential gezogen und hat keinen undefinierten Zustand.

Als Tastereingang wird der Pin 7 genutzt. Pin 11 wird für die Ansteuerung Von Relais 3 genutzt, welches wiederum das grüne Leuchtmittel ansteuert.
Relais 2 schaltet das gelbe Leuchtmittel an und wird von Pin 13 des Raspberry Pis angesteuert. Das rote Leuchtmittel wird von Relais 1 geschaltet. Diese wird vom Pin 15 angesteuert.

6.5 Signierung

Für die Signierung der Challenge wird ein MD5-Hashverfahren sowie ein RSA-Signaturverfahren genutzt. Für die RSA Signierung und die Hashwertbildung wird die mbedTLS[17] Bibliothek genutzt.
Die für die Signierung benötigten Schlüssel wurden zuvor mit einem Key Generator erzeugt und in Dateien abgespeichert. Die Software für die Authentifizierung lädt aus diesen Dateien die Schlüssel und speichert diese auf Variablen innerhalb der Software ab.

6.5.1 mbedTLS

Die mbedTLS ist eine Bibliothek für Transport Layer Security[18], welche speziell für eingebettete Geräte eingesetzt wird, da sie nur sehr wenig Speicherplatz benötigt [22]. Zudem ist die Bibliothek einfach aufgebaut und somit leicht zu verstehen. mbedTLS bietet sehr viele Beispiele und eine sehr ausführliche Dokumentation.

Als alternative hätte OpenSSL[19] genutzt werden können. Diese ist sehr umfangreich und komplex. Das hätte einen größeren Einarbeitungsaufwand zu Folge gehabt. Für den im Rahmen dieser Arbeit gebauten Demonstrator ist die mbedTLS ausreichend.

6.5.2 Schlüssel

Im Rahmen dieser Arbeit wurden die Schlüssel für die Signierung mit einer separaten Software generiert und in Dateien abgespeichert. Für den Demonstrator ist dies zunächst ausreichend. Im weiteren Verlauf müssten die Schlüssel jedoch an einem sicheren Ort abgespeichert werden.

Die generierten Schlüsselpaare werden in folgenden Dateien abgespeichert.

- rsa_priv_OBU: Privater Schlüssel der OBU, zum Signieren der Challenge
- rsa_pub_OBU: Öffentlicher Schlüssel der OBU, zum Verifizieren der Signatur.
- rsa_priv_RSU: Privater Schlüssel der OBU, zum Signieren der Challenge
- rsa_pub_RSU: Öffentlicher Schlüssel der OBU, zum Verifizieren der Signatur.

Die Cohda MK5 verfügt über einen Security Controller IC, dem SmartMX2[20]. Der SmartMX2 ist für solche Anwendungsfälle entwickelt worden und verfügt über kryptographische Funktionen in Hardware. Daher wäre es sinnvoller diesen für die Signierung und Schlüsselaufbewahrung zu nutzen. Das würde jedoch den Rahmen einer Bachelorarbeit übersteigen.

Im Rahmen von iKoPA wird die Signierung der Challenge nicht mehr von der OBU getätigt. Stattdessen übernimmt das Smartphone die Signierung, sodass die Authentifizierungssoftware an sich keine kryptographischen Funktionen mehr benötigt.

[17] https://tls.mbed.org/
[18] Verschlüsselungsprotokoll zur sicheren Datenübertragung [41].
[19] https://www.openssl.org/
[20] http://www.nxp.com/products/identification-and-security/security-controller-ics/smartmx-controllers:MC_53567

6.6 Demoaufbau

Der Demonstrator besteht aus den in Kapitel 6.1 und 6.2 beschrieben Zugangspunkt und Fahrzeug.

Beim Zugangspunkt wurde die Cohda MK5 so konfiguriert, dass diese die AuthApp Software automatisch startet. Auch der Raspberry Pi startet die GpioMK5 Software automatisch beim Hochfahren.

Beim Fahrzeug wird die GpioMK5 Software auf dem Raspberry Pi auch automatisch gestartet. Die AuthApp Software auf der Cohda MK5 wird nicht automatisch gestartet, da das Tablet zunächst eine Verbindung zur Cohda MK5 aufbauen muss, um die Ausgaben auf dem Tablet zu erhalten.
Daher wird die AuthApp-Software über eine Konsoleneingabe auf dem Tablet gestartet.
Über die App Termius wird einen SSH-Verbindung zur Cohda MK5 aufgebaut.

Über den Befehl ***„cd /mnt/ubi/AuthApp“*** wird in das Verzeichnis gewechselt, indem die AuthApp-Software gespeichert ist.
Mit dem Befehl ***„./rc.AuthAppV1 start raw“*** wird die Software gestartet.
Mit dem Befehl ***„./rc.AuthAppV1 stop“*** wird die Software beendet.
Beim Beenden von AuthApp wird auch die GpioMK5 Software auf dem Raspberry Pi beendet.

In den nachfolgenden Codeausschnitten werden mitgeloggte Ausgaben beispielhaft gezeigt. Es werden Ausgaben beim Start und zum Ende einer Authentifizierung gezeigt. Zudem werden die Ausgaben beim empfangen und senden einer Nachricht beispielhaft gezeigt. Die komplette Log-Datei ist im Anhang A: AuthApp_Log.txt angehängt.

6.6.1 AuthApp Ausgabe - Initialisierung

```
24      ------CASE_Start------
25      contacting 192.168.42.129 on port 51717
26
27      GpioRX: 1500
28      GpioRX: 222
29      GpioRX: 0
30      Private Key loaded            [X]
31      Public Key loaded             [X]
32      Search StationID is           1011
33
34      ------CASE_ReadyForAuth------
35
36      WaitForCarTrigger
```

Code 6.11 - Fahrzeugausgabe nach dem Start [Anhang A: AuthApp_Log.txt]

Nach dem Starten der Software wird, wie in Code 6.11 zu sehen, ausgegeben, dass eine Verbindung zur IP-Adresse des Raspberry Pi aufgebaut wird. Wenn dies erfolgreich war, wird die WAITTIME und die Testsequenz der LEDs gesendet. Dadurch das die GpioMK5 Software jeden Befehl bestätigt wird dies auch ausgegeben.

Danach werden die Schlüssel geladen, sowie alle Variablen initialisiert. Nach erfolgreichem Laden der Schlüssel wird dies durch „[X]“ angezeigt. Zudem wird die ID der Ziel-RSU angezeigt. Danach wird durch das anzeigen von „WaitForCarTrigger“, ausgesagt, dass auf das Triggersignal gewartet wird.

6.6.2 AuthApp Ausgabe - Empfangende Nachricht

```
34      ------CASE_ReadyForAuth------
35
36      WaitForCarTrigger
37
38      Receive Message:
39      <RAW>
40          <header>
41              <protocolVersion>1</protocolVersion>
42              <messageID>100</messageID>
43              <stationID>1011</stationID>
44          </header>
45          <auth>
46              <opcode>1</opcode>
47              <data>
48                  <nothing><true/></nothing>
49              </data>
50          </auth>
51      </RAW>
52
53      Receive AuthMsg              [X]
54      Target Station               [X]
55      Receive StationID            1011
```

Code 6.12 - Fahrzeugausgabe Empfangende Nachricht [Anhang A: AuthApp_Log.txt]

Nach dem betätigen des Tasters, werden die ankommenden Nachrichten ausgewertet und angezeigt. Im Code 6.12 - Z.38ff sieht man die Ausgaben empfangende Auth-Nachricht. Darunter befindet sich eine Auswertung dieser. Man erkennt, dass eine Auth-Nachricht empfangen wurde und dass es sich bei dem Sender um die Ziel-Station handelt, anhand der „[X]“ Markierung in der jeweiligen Zeile.

6.6.3 AuthApp Ausgabe - Gesendete Nachricht

```
57      ------CASE_StartAuth------
58      Send CryptoAlgoType:         NoCrypto (0)
59      Mutual Auth:                 Yes (1)
60
61      Send Message:
62      <RAW>
63          <header>
64              <protocolVersion>1</protocolVersion>
65              <messageID>100</messageID>
66              <stationID>5115</stationID>
67          </header>
68          <auth>
69              <opcode>3</opcode>
70              <data>
71                  <cryptoAlgoType>0</cryptoAlgoType>
72              </data>
73          </auth>
74      </RAW>
```

Code 6.13 - Ausgabe - Gesendete Nachricht [Anhang A: AuthApp_Log.txt]

In Code 6.13 ist die Ausgabe beim Senden einer Auth-Nachricht gezeigt. Zunächst wird ausgegeben, welche Daten zum Senden genutzt werden. Im Beispiel im Code 6.13 wird als „Send CryptoAlgoType:“ „NoCrypto“ ausgegeben. Das heißt, dass keine Verschlüsselung für die nachfolgenden Nachrichten verwendet wird. Bei „Mutual Auth:“ wird ein „Yes“ ausgeben, dies bedeutet, dass eine gegenseitige Authentifizierung gestartet wird. Danach wird die gesendete Nachricht ausgegeben.

6.6.4 AuthApp Ausgabe - Ende

Zum Ende einer Authentifizierung wird auf der Fahrzeugseite auf eine Zugangsbestätigung bzw. Zugangsverweigerung gewartet.
Bei erfolgreicher Authentifizierung und gewährtem Zugang wird ein „Weg get access“ mit einem symbolischen Abhacken ausgegeben [Code 6.15].

```
318       We get access
319
320     |================|
321     |              X |
322     |            X   |
323     | X        X     |
324     |   X    X       |
325     |     X          |
326     |================|
327
328     ------CASE_End------
329
330     ------CASE_ReadyForAuth-----
331
332     WaitForCarTrigger
```

Code 6.15 - Ausgabe Zugang gewährt [Anhang A: AuthApp_Log.txt]

```
318       We get NO access
319
320     |================|
321     |   X        X   |
322     |     X    X     |
323     |       X        |
324     |     X    X     |
325     |   X        X   |
326     |================|
327
328     ------CASE_End------
329
330     ------CASE_ReadyForAuth-----
331
332     WaitForCarTrigger
```

Code 6.14 - Ausgabe Zugang verweigert [Anhang A: AuthApp_Log.txt]

Bei fehlgeschlagener Authentifizierung und verweigertem Zugang wird auf dem Fahrzeug ein symbolisches X dargestellt [Code 6.14].

Zum Schluss wird angezeigt, dass in den Zustand „End“ gewechselt wurde und die Authentifizierung beendet wurde. Danach wird wieder in den Zustand „ReadyForAuth“ gewechselt und auf ein neues Triggersignal gewartet.

In Abbildung 6.13 ist der funktionierende Demonstrator gezeigt.

Abbildung 6.13 - Funktionierender Demonstrator [25]

7 Bewertung und Ausblick

In diesem Kapitel wird die Arbeit vor dem Hintergrund der Ausarbeitung und des Demonstrators zusammengefasst dargestellt.
Es wird ein Fazit über die Arbeit getroffen. Im Abschnitt Ausblick wird dargestellt, welche weiteren zukünftige Forschungsmöglichkeiten durch diese Ausarbeitung gegeben sind.

7.1 Zusammenfassung

In dieser Arbeit wurde für Fahrzeuge eine Authentifizierungsmöglichkeit über ITS-G5 entwickelt.
Es wurden in Kapitel 4 die Anforderung, des Förderprojektes iKoPA, der Standardisierung und von NXP analysiert.

Es zeigte sich, dass nicht alle Anforderung, wie beschrieben zum aktuellen Zeitpunkt umsetzbar waren. So ist eine Anforderung in iKoPA, dass ein Smartphone die Signierung für das Authentifizierungsverfahren übernimmt. Aktuell stand noch kein Smartphone zur Verfügung. Daher wird die Challenge in dieser Arbeit von der OBU signiert. Das Verarbeiten der Challenge wurde in der Software sehr modular implementiert, sodass zu einem späteren Zeitpunkt die Signierung ausgelagert bzw. durch das Weiterleiten an ein Smartphone ersetzt werden kann.

Des Weiteren ist die Ansteuerung eines optischen Signalgebers bzw. einer Schranke nicht durch die Cohda MK5 OBU direkt möglich, da diese über keine GPIO-Ausgänge verfügt, welche im Rahmen dieser Arbeit genutzt werden können. Es wäre Möglich die MK5 zu modifizieren und GPIO-Ausgänge herauszuführen. Dies wäre mit einem erheblichen Risiko und Aufwand verbunden. Daher wurde für die Signalansteuerung ein alternatives Konzept, auf Basis eines Raspberry Pis mit Relais-Board, entwickelt.

Aufbauend auf diesen Anforderungen wurde im Kapitel 5 ein Konzept für eine Authentifizierungsmöglichkeit, sowie einen Demonstrator erarbeitet.

Es wurde eine neue Nachricht für die Authentifizierung über ITS-G5 entwickelt. Diese Nachricht wurde im Rahmen dieser Arbeit „Auth-Message“ genannt. Sie ist ähnlich einer CAM aufgebaut und wurde in ASN.1 erarbeitet. Dadurch ist die Möglichkeit eine Standardisierung dieser Nachricht durch die Industrie gegeben.

Das Konzept wurde durch die Möglichkeit einer gegenseitigen Authentifizierung erweitert. So ist es möglich, dass sich nicht nur das Fahrzeug an dem Zugangspunkt authentifiziert, sondern auch der Zugangspunkt an dem Fahrzeug. Dadurch kann eine gewisse Vertrauensbasis geschaffen werden.

Die Entwicklung und Umsetzung dieses Konzeptes wird in Kapitel 6 dargestellt und erläutert. Durch die dargestellten Ausgaben und Abbildungen wird gezeigt, dass die Entwicklung die Anforderungen erfüllt.

Der Demonstrator zeigt die Funktion des entwickelten Authentifizierungsverfahren über ITS-G5. Zudem ist ein Tablet in den Aufbau integriert, welche die Abläufe während der Authentifizierung für den Nutzer transparent darstellt.

7.2 Fazit

Diese Arbeit zeigt, dass eine Authentifizierung mit einer neuen Nachrichtenform über ITS-G5 möglich ist.

Durch die Implementierung der Auth-Message in ASN.1 kann diese für jedes V2X-fähige Gerät genutzt werden. Es sind verschiedenste Einsatzmöglichkeiten dieser Nachricht denkbar. Wie in dem Demonstrator gezeigt, könnte mit Hilfe der Auth-Message eine effiziente Zugangskontrolle stattfinden. Die Nachricht könnte auch genutzt werden um geheime Information zwischen zwei Partner auszutauschen. Dafür wurde die Entwicklung um die gegenseitige Authentifizierung erweitert. Zudem wurde die Möglichkeit einer Verschlüsselung der Nachrichten vorgesehen und mit implementiert. Lediglich die Verschlüsselungsalgorithmen müssten ergänzt werden.

Im Rahmen dieser Arbeit wurde die Software auf Basis der bestehenden „ExampleETSI“ Software von Cohda Wireless entwickelt, da eine Neuentwicklung den Rahmen dieser Arbeit überschritten hätte. Dies hat zufolge, dass die „Auth-Message“ nicht über eine direkte Verbindung zum Kommunikationspartner übertragen wird, sondern durch das Geonetworking Protokoll an alle sich in der Nähe befindlichen V2X-fähigen Geräte gesendet wird.

Dadurch, dass andere Partner die Schlüssel zur Signierung und Verifizierung nicht kennen, ist dies kein Problem. Jedoch würde jeder in Nähe befindlicher Angreifer alle gesendeten Nachrichten auch empfangen. Dadurch wäre eine Verfolgung des Nutzers möglich. Daher sollte die Kommunikation besser auf einem direkten Weg, wie TCP/IP, erfolgen.

Für das Signaturverfahren werden Schlüsselpaare benötigt. Diese werden im Rahmen dieser Arbeit vorher erzeugt und lokal in der Ordnerstruktur der Software abgespeichert.
Dies ist kein sicherer Ort für die Aufbewahrung solcher Schlüssel. Die Cohda MK5 verfügt über den SmartMX2 Security Controller IC, welcher in dieser Arbeit nicht betrachtet wurde. Es ist sinnvoll die gesamte Kryptografie auf diesen IC auszulagern.

Durch den modularen Aufbau der Software kann diese erweitert und auch in andere Projekte mit implementiert werden.

7.3 Ausblick

Es geht hervor, dass die entwickelte Software noch gewisse Schwächen aufweist, welche im Rahmen dieser Arbeit nicht behoben werden konnten.

Ein nächster Schritt wäre es die Software von dem Geonetworking Protokoll auf ein direktes Protokoll umzustellen, sowie die kryptografischen Funktionen auf den SmartMX2 auszulagern.

Für NXP wird der Demonstrator weiterentwickelt. Es wird eine Vernetzung zwischen dem RFID-Lesegerät und der OBU realisiert. Zudem wird die NFC-Technologie mit in den Demonstrator integriert. Auch wird eine Anbindung an einen Server geschaffen, welcher für die Schlüsselgenerierung und Aufbewahrung dieser verantwortlich sein wird.

Für iKoPA wird die Software dahingehend angepasst, dass die Generierung und Aufbewahrung der Schlüssel beim Registrierungsservice liegt. Die Signierung wird auf das Smartphone ausgelagert.

Die Auth-Message kann als Grundlage genutzt werden um zum Beispiel einen Bezahldienst oder einen gesicherten Informationsaustausch über ITS-G5 zu realisieren.
Durch NXP oder andere Partner kann die Standardisierung der Auth-Message vorangetrieben werden, sodass diese von allen V2X-fähigen Geräten wie eine CAM oder DENM genutzt werden kann.

Durch diese Arbeit ist die Grundlage für eine Authentifizierungsmöglichkeit über ITS-G5 geschaffen worden, auf welcher weiter aufgebaut werden kann, um in Zukunft Dienste für ITS-G5 zu schaffen, welche eine Authentifizierung benötigen.

Literaturverzeichnis

[1] „tu-ilmenau,“ [Online]. Available: https://www.tu-ilmenau.de/fileadmin/media/telematik/schmidt/080718_V2X-Kommunikation.pdf. [Zugriff am 14 Januar 2017].

[2] „WinfWiki,“ [Online]. Available: http://winfwiki.wi-fom.de/index.php/Standards_in_der_Car-To-Car-Kommunikation. [Zugriff am 12 Januar 2017].

[3] „OSS ASN1,“ [Online]. Available: http://www.oss.com/asn1/resources/development-with-asn1/asn1-development-workflow.html. [Zugriff am 17 Januar 2017].

[4] „ASN.1 Dokumentation,“ [Online]. Available: https://www.obj-sys.com/asn1tutorial/node10.html. [Zugriff am 17 Janaur 2017].

[5] „ISO/OSI-7-Schichtenmodell,“ [Online]. Available: http://www.elektronik-kompendium.de/sites/kom/0301201.html. [Zugriff am 29 Januar 2017].

[6] Rohde & Schwarz, „Intelligente Verkehrssysteme mit IEEE 802.11p,“ [Online]. Available: https://cdn.rohde-schwarz.com/pws/dl_downloads/dl_application/application_notes/1ma152/1MA152_4e_ITS_using_802_11p.pdf. [Zugriff am 20 Januar 2017].

[7] D.-I. M. R. Univ.-Prof. Dr. Thomas Strang, „STI-Innsbruck,“ [Online]. Available: http://www.sti-innsbruck.at/sites/default/files/courses/fileadmin/documents/vn-ws0809/01-vn-intro.pdf. [Zugriff am 17 Januar 2017].

[8] „1609.4 - IEEE Standard for Wireless Access in Vehicular Environments (WAVE) -- Multi-Channel Operation,“ [Online]. Available: http://ieeexplore.ieee.org/document/7553418/. [Zugriff am 20 Januar 2017].

[9] „1609.3 - IEEE Standard for Wireless Access in Vehicular Environments (WAVE) -- Networking Services,“ [Online]. Available: http://ieeexplore.ieee.org/document/7458115/. [Zugriff am 20 Januar 2017].

[10] „1609.2 - IEEE Standard for Wireless Access in Vehicular Environments--Security Services for Applications and,“ [Online]. Available: http://ieeexplore.ieee.org/document/7426684/. [Zugriff am 15 Januar 2017].

[11] ETSI, „Intelligent Transport Systems (ITS);,“ [Online]. Available: http://www.etsi.org/deliver/etsi_en/302600_302699/302665/01.01.01_60/en_302665v010101p.pdf. [Zugriff am 2017 Januar 27].

[12] ETSI, „ITS CAM,“ [Online]. Available: http://www.etsi.org/deliver/etsi_ts/102600_102699/10263702/01.02.01_60/ts_10263702v010201p.pdf. [Zugriff am 05 Feburar 2017].

[13] ETSI, „ITS DENM," [Online]. Available: http://www.etsi.org/deliver/etsi_ts/102600_102699/10263703/01.01.01_60/ts_10263703v010101p.pdf. [Zugriff am 05 Feburar 2017].

[14] A. M. Klingler, „Authentifizierungsverfahren und ihre Benutzerfreundlichkeit," 2011. [Online]. Available: https://www.secuso.informatik.tu-darmstadt.de/fileadmin/user_upload/Group_SECUSO/Theses/BA/Andreas_Marc_Klingler_BA.pdf. [Zugriff am 24 Januar 2017].

[15] „Kryptografische Hash-Funktionen," [Online]. Available: http://www.elektronik-kompendium.de/sites/net/1909041.htm. [Zugriff am 30 Januar 2017].

[16] H. Y. Xiaoyun Wang, „How to Break MD5 and Other Hash Functions," [Online]. Available: http://download.springer.com/static/pdf/55/chp%253A10.1007%252F11426639_2.pdf?originUrl=http%3A%2F%2Flink.springer.com%2Fchapter%2F10.1007%2F11426639_2&token2=exp=1488970870~acl=%2Fstatic%2Fpdf%2F55%2Fchp%25253A10.1007%25252F11426639_2.pdf%3ForiginUrl%3Dh. [Zugriff am 08 März 2017].

[17] C. Stachniss, „Uni Freiburg - Kryptographische Hashfunktionen," [Online]. Available: http://www2.informatik.uni-freiburg.de/~stachnis/pdf/stachniss-habil-talk-hashfunctions.pdf. [Zugriff am 08 März 2017].

[18] P. D. W. Riggert, „Digitale Signatur FH Flensburg," [Online]. Available: http://www.wi.fh-flensburg.de/fileadmin/dozenten/Riggert/bildmaterial/Dokumentenmanagement/4-Deliver-Digitale-Signatur.pdf. [Zugriff am 25 Januar 2017].

[19] P. v. O. a. S. V. A. Menezes, Handbook of Applied Cryptography, CRC Press, 1996.

[20] „NXP Road Link," [Online]. Available: http://www.nxp.com/products/automotive-products/advanced-automotive-safety/roadlink-technology:ROADLINK-TECH. [Zugriff am 15 Februar 2017].

[21] „Cohda Wireless Support," [Online]. Available: https://support.cohdawireless.com/hc/en-us. [Zugriff am 09 Februar 2017].

[22] „mbedTLS," [Online]. Available: https://tls.mbed.org/. [Zugriff am 10 Feburar 2017].

[23] ETSI, „ETSI BTP," [Online]. Available: http://www.etsi.org/deliver/etsi_en/302600_302699/3026360501/01.02.01_60/en_3026360501v010201p.pdf. [Zugriff am 20 01 2017].

[24] L. Mustafa, *Selber erstellte Abbildung mit Enterprise Architect (SparxSystems),* Hamburg, 2017.

[25] L. Mustafa, *Selber erstellte Abbildung mit Powerpoint (Office 16),* Hamburg, 2017.

[26] „Ad-hoc-Netz," [Online]. Available: https://de.wikipedia.org/wiki/Ad-hoc-Netz. [Zugriff am 26. Februar 2017].

[27] „Authentifizierung," [Online]. Available: https://de.wikipedia.org/wiki/Authentifizierung. [Zugriff am 25 Januar 2017].

[28] Cohda Wireless, „Cohda Wireless MK5," [Online]. Available: http://cohdawireless.com/Portals/0/PDFs/CohdaWirelessMK5.pdf. [Zugriff am 27 Januar 2017].

[29] „Compiler," [Online]. Available: https://de.wikipedia.org/wiki/Compiler. [Zugriff am 27 Februar 2017].

[30] „Fakra - Wikipedia,“ [Online]. Available: https://de.wikipedia.org/wiki/Koaxialstecker#SMBA-.28FAKRA.29-Steckverbinder. [Zugriff am 01 März 2017].

[31] „Hashwert,“ [Online]. Available: http://www.itwissen.info/definition/lexikon/Hashwert-hash-value.html. [Zugriff am 12 März 2017].

[32] htw Saar, „htw Saar,“ [Online]. Available: https://fgvt.htwsaar.de/public/index.php/ikopa-2015-2018/. [Zugriff am 05 Februar 2017].

[33] „Integer,“ [Online]. Available: https://de.wikipedia.org/wiki/Integer_(Datentyp). [Zugriff am 27 Feburar 2017].

[34] „NXP A700X,“ [Online]. Available: http://www.nxp.com/products/identification-and-security/secure-authentication-and-anti-counterfeit-technology/secure-authentication-microcontroller:A700X_FAMILY. [Zugriff am 01 März 2017].

[35] „NXP UCODE DNA,“ [Online]. Available: http://www.nxp.com/products/identification-and-security/smart-label-and-tag-ics/ucode-dna:MC_1436185909808. [Zugriff am 08 März 2017].

[36] „Optokoppler - Wikipedia,“ [Online]. Available: https://de.wikipedia.org/wiki/Optokoppler. [Zugriff am 01 März 2017].

[37] „Performance evaluation of IEEE 1609 WAVE and IEEE 802.11p for vehicular communications,“ [Online]. Available: http://ieeexplore.ieee.org/document/5547184/. [Zugriff am 15 Januar 2017].

[38] „Protokollstack,“ [Online]. Available: http://www.itwissen.info/definition/lexikon/Protokollstack-protocol-stack.html. [Zugriff am 27 Februar 2017].

[39] NXP Semiconductors, „SAF5100 (Firmenintern),“ 2014.

[40] Swarco, „Swarco Futura,“ [Online]. Available: https://www.swarco.com/futurit/content/download/14749/215480/file/FUTURA%20Produktfolder.pdf. [Zugriff am 25 Februar 2017].

[41] „TLS - Wikipedia,“ [Online]. Available: https://de.wikipedia.org/wiki/Transport_Layer_Security. [Zugriff am 01 März 2017].

[42] „Wikipedai - Debugger,“ [Online]. Available: https://de.wikipedia.org/wiki/Debugger. [Zugriff am 10 März 2017].

[43] „Wikipedia - Gateway,“ [Online]. Available: https://de.wikipedia.org/wiki/Gateway_(Informatik). [Zugriff am 02 März 2017].

[44] „Wikipedia AES128,“ [Online]. Available: https://de.wikipedia.org/wiki/Advanced_Encryption_Standard. [Zugriff am 01 März 2017].

[45] K. Sjöberg, „Högskolan Halmstad, WAVE IEEE ETSI,“ [Online]. Available: http://www.hh.se/download/18.16490bf0133abd6557e8000209/1341267475825/WWVC2011Standardization.KS.pdf. [Zugriff am 10 Februar 2017].

Anhang

Folgende Dateien und Dokumente befinden sich auf der CD im Ordner Anhang.
(Alle Codedateien wurden, für das Betrachten, zusätzlich als rtf-Datei abgespeichert.)

A Code

- AuthApp
  - RAW.asn1
  - Raw.cfg
  - Raw.conf
  - Raw-its.c
  - Raw-its.h
- CohdaMK5GPIO
  - Main_GPIOMK5.c
- Dokumentation
  - Hier sind alle Codeausschnitte aus der Arbeit in einzelnen Dateien abgelegt
- Wireshark_Logs
  - Screenshots von einzelnen Nachrichten aus Wireshark
- AuthApp_Log.txt

B Software

- AuthApp
- CohdaMK5GPIO

C Dokumente

- Authentifikation_Andreas_Marc_Klingler_BA.pdf
- Bruce_Schneier_Sensible authentication.pdf
- CohdaWirelessIntro_Handout
- CohdaWirelessMK5
- ETSI CAM en 30263702v010301v.pdf
- ETSI DENM ts 10263703v010101p.pdf
- ETSI EN 302 663.pdf
- ETSI EN 302 665.pdf
- ETSI_UseCases_ts_10263701v010101p.pdf
- Frequenzband _802_11p_IEEE.pdf
- HowtoBreakMD5andOtherHashFunctions.pdf
- IEEE 1609.2.pdf
- IEEE 1609.3.pdf
- IEEE 1609.4.pdf
- IEEE802_11pStandard.pdf
- Rohde Schwarz ITS 802.11p.pdf
- stachniss-habil-talk-hashfunctions.pdf
- SwarcoFUTURA Produktfolder.pdf
- V2X_Communication_Protocols_DLR.pdf

Versicherung über Selbstständigkeit

Hiermit versichere ich, dass ich die vorliegende Arbeit ohne fremde Hilfe selbstständig verfasst und nur die angegebenen Hilfsmittel benutzt habe.

Hamburg, den _____ _____