

# Inhaltsverzeichnis

<b>1 Einleitung</b>	<b>1</b>
1.1 Motivation . . . . .	1
1.2 Problemstellung und Abgrenzung . . . . .	1
1.3 Ziel der Arbeit . . . . .	2
1.4 Vorgehen . . . . .	3
<b>2 Fahrzeugnetzwerke</b>	<b>4</b>
2.1 Historie . . . . .	4
2.2 Konzepte und Anwendungsgebiete von Bussystemen . . . . .	5
2.2.1 On-Board-Kommunikation . . . . .	6
2.2.2 Off-Board-Kommunikation . . . . .	7
2.3 Übersicht der Busprotokolle . . . . .	8
2.3.1 K-Leitung / K-Line mit KWP2000 . . . . .	12
2.3.2 CAN . . . . .	15
2.3.3 LIN . . . . .	22
2.4 KWP2000 . . . . .	24
2.4.1 UDS . . . . .	27
2.5 Schnittstellen von Automobilnetzwerken . . . . .	28
2.5.1 OBD-II . . . . .	28
2.5.2 Abzweigung von Busleitungen . . . . .	30
<b>3 Analyse von Fahrzeugnetzwerken</b>	<b>33</b>
3.1 Vorbereitung . . . . .	33
3.1.1 Hardware . . . . .	33
3.1.2 Software . . . . .	37
3.2 Aufbau von Testumgebungen . . . . .	40
3.3 Virtuelle Testumgebungen . . . . .	41
3.4 Physische Testumgebungen . . . . .	43
3.5 Angriffsmethoden und Techniken des Reverse-Engineerings . . . . .	45
3.5.1 Manuelle Requests . . . . .	45
3.5.2 Fuzzing . . . . .	47
3.5.3 Dump-Erstellung und Replay-Attacken . . . . .	48
3.5.4 Angriffe auf UDS . . . . .	49
3.5.5 Man-In-The-Middle-Angriffe . . . . .	52
3.5.6 Reverse-Engineering von Hersteller-Software . . . . .	52

<b>4 Entwicklung einer Analysesoftware</b>	<b>54</b>
4.1 Grundlegende Informationen . . . . .	54
4.1.1 Funktionsumfang . . . . .	55
4.1.2 Verwendete Bibliotheken und Pakete . . . . .	56
4.1.3 Architektur . . . . .	56
4.2 Implementierungsdetails . . . . .	57
4.2.1 Prozess der Datenaufzeichnung . . . . .	57
4.2.2 FilterTab: Automatische Paketfilterung . . . . .	59
4.2.3 SearcherTab: Binäre Paketsuche . . . . .	59
4.2.4 Datenbank . . . . .	60
4.2.5 Tab-Vererbung . . . . .	62
<b>5 Analyseergebnisse</b>	<b>63</b>
5.1 Reverse-Engineering eines Kombiinstruments . . . . .	63
5.1.1 Reverse-Engineering der Pinbelegung . . . . .	63
5.1.2 Reverse-Engineering der CAN-Matrix . . . . .	64
5.1.3 Interaktion mit Diagnosediensten . . . . .	65
5.2 Reverse-Engineering von Wartungssoftware . . . . .	67
5.2.1 REDACTED . . . . .	67
5.2.2 REDACTED . . . . .	70
5.3 Weitere Ergebnisse . . . . .	70
5.3.1 Einsatz von Prüfsummen . . . . .	71
5.3.2 Einleiten vom Bremsvorgängen . . . . .	71
5.3.3 Aufrechterhaltung der Beschleunigung . . . . .	71
5.3.4 Eingriff in elektronische Lenkungen . . . . .	72
5.3.5 Statischer UDS-Seed . . . . .	72
<b>6 Aufgetretene Probleme</b>	<b>74</b>
6.1 Protokollspezifikationen . . . . .	74
6.2 Buszugriff und Businteraktion . . . . .	74
6.3 Optimierung der Analysesoftware . . . . .	74
6.4 Analyse von proprietären CAN-Geräten . . . . .	75
<b>7 Zusammenfassung</b>	<b>76</b>
<b>8 Ausblick</b>	<b>78</b>

# Abbildungsverzeichnis

1	Fahrzeugnetzwerke: Bustopologie . . . . .	4
2	Fahrzeugnetzwerke: Typisches Bussystem moderner Fahrzeuge (stark vereinfacht) . . . . .	6
3	Fahrzeugprotokolle: Diagnosevorgang . . . . .	11
4	Fahrzeugprotokolle: Kosten- und Erscheinungsübersicht . . . . .	12
5	Fahrzeugprotokolle: KWP2000: Paketformat auf Busschicht . . . . .	14
6	Fahrzeugprotokolle: K-Line: Paketformat (nach ISO 9141-2) . . . . .	15
7	Fahrzeugprotokolle: CAN: Schematischer Aufbau (High-Speed-CAN) . . . . .	16
8	Fahrzeugprotokolle: CAN-Paketformat . . . . .	17
9	Fahrzeugprotokolle: CAN: Vergleich von Standard- und Extended-Frames . . . . .	18
10	Fahrzeugprotokolle: ISO-TP: Paketformate . . . . .	19
11	Fahrzeugprotokolle: TP 2.0: Broadcast-Nachrichten . . . . .	20
12	Fahrzeugprotokolle: TP1.6/2.0: Paketaufbau . . . . .	21
13	Fahrzeugprotokolle: CAN-FD: Paketaufbau . . . . .	22
14	Fahrzeugprotokolle: LIN: Busaufbau . . . . .	23
15	Fahrzeugprotokolle: LIN: Paketformat . . . . .	23
16	Fahrzeugprotokolle: KWP2000: Paketformat auf Anwendungsschicht . . . . .	24
17	Fahrzeugprotokolle: UDS: Paketformat . . . . .	27
18	OBD-II: Typen von Steckverbindern . . . . .	29
19	OBD-II: Pinbelegungen . . . . .	29
20	OBD-II: Herstellerspezifische Pinbelegungen . . . . .	30
21	Netzwerkschnittstellen: Lokalisierung von CAN-Leitungen . . . . .	31
22	Hardware: USB2CAN . . . . .	33
23	Hardware: <i>USBtin</i> und <i>CANtact</i> . . . . .	34
24	Hardware: OBD-Kabeltypen . . . . .	36
25	Software: CAN-Sniffing mit <i>Wireshark</i> . . . . .	40
26	Software: ICSim . . . . .	42
27	Analyse: Aufbau einer physischen Testumgebung . . . . .	44
28	Analyse: Anschlusspins des Kombiinstruments . . . . .	44
29	Analyse: Physische Testumgebung . . . . .	45
30	Reverse-Engineering: Binäre Paketsuche . . . . .	49
31	Analyse von Fahrzeugnetzwerken: Forenbeitrag eines Schlüsselverkäufers . . . . .	51
32	Analysesoftware: CANalyzat0r: Main-Tab . . . . .	55

33	Analysesoftware: CANalyzat0r: Software-Architektur (vereinfacht) . . . . .	57
34	Analysesoftware: Prozess der Datenaufzeichnung . . . . .	58
35	Analysesoftware: Datenbankschema . . . . .	60
36	Ergebnisse Kombiinstrument: Reverse-Engineering der Pinbelegung . . . . .	63
37	Ergebnisse Kombiinstrument: UDS-Anfrage in Verbindung mit CAN . . . . .	66
38	Ergebnisse Kombiinstrument: UDS-Antwort des Kombiinstruments . . . . .	66
39	Ergebnisse IDA: Teil des Schlüssel-Arrays der Diagnosesoftware . . . . .	68
40	Ergebnisse IDA: Debug-Statement und Aufruf des Schlüsselalgorithmus . .	69

# Tabellenverzeichnis

1	Fahrzeugprotokolle: OSI-Modell im Bezug auf Fahrzeugnetzwerke . . . . .	8
2	Fahrzeugprotokolle: Geschwindigkeitsklassen . . . . .	9
3	Fahrzeugprotokolle: Protokollübersicht . . . . .	10
4	Fahrzeugprotokolle: KWP2000: Service-IDs . . . . .	25
5	Fahrzeugprotokolle: KWP2000: Session-Modi . . . . .	25
6	OBD-II: Pinbelegungen: Legende . . . . .	29
7	Software: Wichtige Komponenten aus can-utils . . . . .	39
8	Reverse-Engineering eines Kombiinstruments: Ermittelte CAN-Matrix (1) .	81
9	Reverse-Engineering eines Kombiinstruments: Ermittelte CAN-Matrix (2) .	82
10	Reverse-Engineering eines Kombiinstruments: Diagnosedienste . . . . .	85
11	Vermutete ECU-Schlüssel (Teil 1 – ASCII-Schlüssel) . . . . .	86
12	Vermutete ECU-Schlüssel (Teil 2) . . . . .	87
13	Vermutete ECU-Schlüssel (Teil 3) . . . . .	88

# Listings

1	Software: Systemlog-Ausgabe bei der Verwendung von USB2CAN . . . . .	38
2	Software: Einstellung der Bitrate . . . . .	38
3	Software: Starten von candump . . . . .	38
4	Software: Ausgabe von candump . . . . .	38
5	Software: Erstellung eines virtuellen CAN-Interfaces . . . . .	41
6	Reverse-Engineering: Manueller Request (Motordrehzahl) . . . . .	46
7	Reverse-Engineering: Diagnoseantwort (Motordrehzahl) . . . . .	46
8	Reverse-Engineering: Fuzzing mit cangen (1) . . . . .	47
9	Reverse-Engineering: Fuzzing mit cangen (2) . . . . .	47
10	Reverse-Engineering: Dump-Erstellung . . . . .	48
11	Reverse-Engineering: Replay-Attacke . . . . .	48
12	Analyse von Fahrzeugnetzwerken: UDS-Heartbeat . . . . .	52
13	Analysesoftware: DatabaseStatements: getInsertStatement . . . . .	60
14	Analysesoftware: DatabaseStatements: getInsertStatement (Rückgabe) . . . . .	61
15	Analysesoftware: DatabaseStatements: getInsertProjectStatement . . . . .	61
16	Analysesoftware: Database: saveProject . . . . .	61
17	Analyseergebnisse: Vermuteter Authentifizierungsalgorithmus für Suzuki-Fahrzeuge . . . . .	69
18	Analyseergebnisse: Unbekannter potentieller Authentifizierungsalgorithmus (1) . . . . .	70
19	Analyseergebnisse: Skript zur Geschwindigkeitskonvertierung . . . . .	83
20	REDACTED . . . . .	88
21	Analyseergebnisse: Unbekannter Authentifizierungsalgorithmus (2) . . . . .	90
22	Analyseergebnisse: Unbekannter Authentifizierungsalgorithmus (2): changeKey . . . . .	90
23	REDACTED . . . . .	91
24	REDACTED . . . . .	

## Akronyme

**ABS** Antiblockiersystem. 4

**BRS** Bit Rate Switch. 22

**CAN** Controller Area Network. 2

**CRC** Cyclic Redundancy Check. 15

**EAP** Extensible Authentication Protocol. 78

**ECU** Electrical Control Unit. 6

**EDL** Extended Data Length. 22

**ESI** Error State Indicator. 22

**GPIO** General Purpose Input/Output. 35

**GUI** Graphical User Interface. 54

**IDE** Identifier Extension. 18

**IDS** Integrated Diagnostic Software. 67

**LIN** Local Interconnect Network. 22

**MAC** Media Access Control. 78

**MITM** Man In The Middle. 37

**MOST** Media Oriented Systems Transport. 7

**OBD** On Board Diagnose. 7

**OSI** Open System Interconnection. 8

**PCI** Protocol Control Information. 19

**PID** Parameter Identifier. 23

**RTR** Remote Transmission Request. 17

**SAE** Society of Automotive Engineers. 22

**SID** Service Identifier. 14

**SRR** Substitute Remote Request. 18

**UDS** Unified Diagnostic Services. 13

**VMM** Vehicle Message Matrix. 16

## 2 Fahrzeugnetzwerke

### 2.1 Historie

Der Weg bis hin zu den heutigen komplexen Bussystemen begann im Jahr 1915, als zum ersten Mal ein elektronisches Bauteil in Form eines Scheinwerfers in Fahrzeugen verbaut wurde [dsc]. Der Ausbau der Vernetzung ging zunächst mäßig vonstatten – im Jahr 1972 beanspruchte der Schaltplan eines *VW Käfer* nach wie vor lediglich eine DIN A4-Seite [dsc] [ksc].

Durch die in den 1970er und 1980er-Jahren entwickelten Mikrocontroller und den damit verbundenen neuen technischen Möglichkeiten erfolgte jedoch eine vermehrte Ausrichtung des Fahrzeugbaus in Richtung Elektronik und Vernetzung: Systeme wie das **Antiblockiersystem (ABS)** fordern zunächst einen Punkt-zu-Punkt-Austausch von Daten zwischen Sensoren und Steuergeräten [Wer14]. Neu entworfene Fahrzeugsysteme werden zunehmend auch in preiswerteren Automobilen implementiert.

Aufgrund von Skalierbarkeit, Kostenminimierung und erleichterter Realisierbarkeit begann die Entwicklung und Verwendung von Bussystemen im Fahrzeughbereich. Unter anderem werden eigens für die Fahrzeugbranche entwickelte Bussysteme spezifiziert, welche den speziellen Anforderungen dieser Branche gerecht werden können.

Bei einem Busnetzwerk sind die Netzwerkteilnehmer in einer Kette hintereinander angeordnet [psc]. Es existiert eine gemeinsame Datenleitung, auf welche alle Teilnehmer gleichermaßen zugriff haben. Für das Management des Datenverkehrs sind alle Teilnehmer zusammen verantwortlich – es gibt keine zentrale Netzwerkinstantz, welche die Paketverwaltung übernimmt. Die Netzwerkknoten bestimmen im Falle von Busnetzen selbst, ob sie eine Nachricht ignorieren und lediglich weiterleiten oder ob sie zusätzlich verarbeitet wird.

Folgendes Schaubild veranschaulicht den Aufbau der Bustopologie:

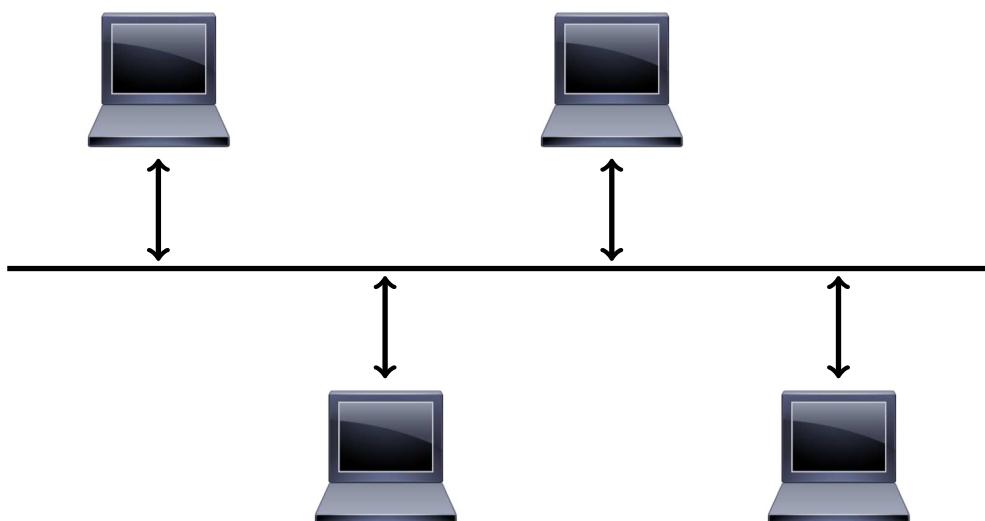


Abbildung 1: Fahrzeugnetzwerke: Bustopologie

Grafiken entnommen aus: [cic]

Bussysteme erlauben einen effizienten Aufbau des Fahrzeugnetzwerks [kun]. Jeder Knoten muss nur einmal verbunden werden. Rohstoffe, Kosten, Platz und Gewicht können somit eingespart werden. Bei der Betrachtung der hohen Stückzahl der Fahrzeuge und der Anzahl der Netzwerkteilnehmer ist die Auswirkung einer effizienten Busplanung ein tragender Preisfaktor. Es besteht zudem die Möglichkeit der Absicherung des Netzes durch redundante Datenleitungen. Die Integration weiterer Teilnehmer und sich ändernde Anforderungen stellen darüber hinaus keinen großen Aufwand dar [kun].

Aufgrund der Buscharakteristik ist es *prinzipiell* möglich den gesamten Netzwerkverkehr zu überwachen oder zu manipulieren, da theoretisch jeder Teilnehmer alle übertragenen Daten erhält. Es können jedoch auch segmentierte Fahrzeugnetzwerke vorkommen [miva]. Viele Protokolle, wie auch das CAN-Bus-Protokoll, implementieren zudem keine Quelladressen oder Authentifizierung für Netzwerkpakete.

Während das Aufzeichnen und Einschleusen von Paketen vergleichsweise einfach zu bewerkstelligen ist, ist eine Datenverkehr-Analyse meist wesentlich schwieriger durchzuführen. Dieser Umstand ist darin begründet, dass in Netzwerkpaketen bei den meisten Protokollen keine identifizierenden Informationen von Sender oder Empfänger enthalten sind. Ohne zusätzliche Maßnahmen kann somit nur selten direkt festgestellt werden, welches Steuergerät der Sender bzw. Empfänger einer Nachricht ist.

Mit der wachsenden Komplexität von Fahrzeugnetzwerken stieg der Bedarf an Diagnosemethoden der Busse. Aufgrund der zunehmenden Anzahl von Aufgabenbereichen, welche von Busnetzwerken übernommen werden, entstehen Netzwerke mit diversen Subnetzen und einer großen Menge an Netzwerknoten. Es sollte anfänglich ein gewisser Grad an Standardisierung erreicht werden, um einheitliche Netzwerke und Diagnosesysteme zu entwickeln. Aus diesem Grund folgten einige Zeit später Standards zur Diagnose und Vernetzung von Netzwerknoten in Fahrzeugen.

Schaltpläne aktueller Automobile beanspruchen heutzutage unzählige Seiten Papier und werden selten in Papierform verwendet, da die Pläne dieser Fahrzeuge ganze Handbücher umfassen.

## 2.2 Konzepte und Anwendungsgebiete von Bussystemen

Grundsätzlich existieren zwei verschiedene Kommunikationskategorien, welche unterschiedliche Aufgaben erfüllen und somit abweichende Einsatzgebiete besitzen [Wer14]. Diese werden in den folgenden Abschnitten näher erläutert.

Nachfolgendes Schaubild (angelehnt an [Wer14] S. 2) bietet eine erste grobe Übersicht eines modernen Bussystems, welches mit der Unterteilung in On-Board und Off-Board-Kommunikation beide erwähnten Gebiete umfasst:

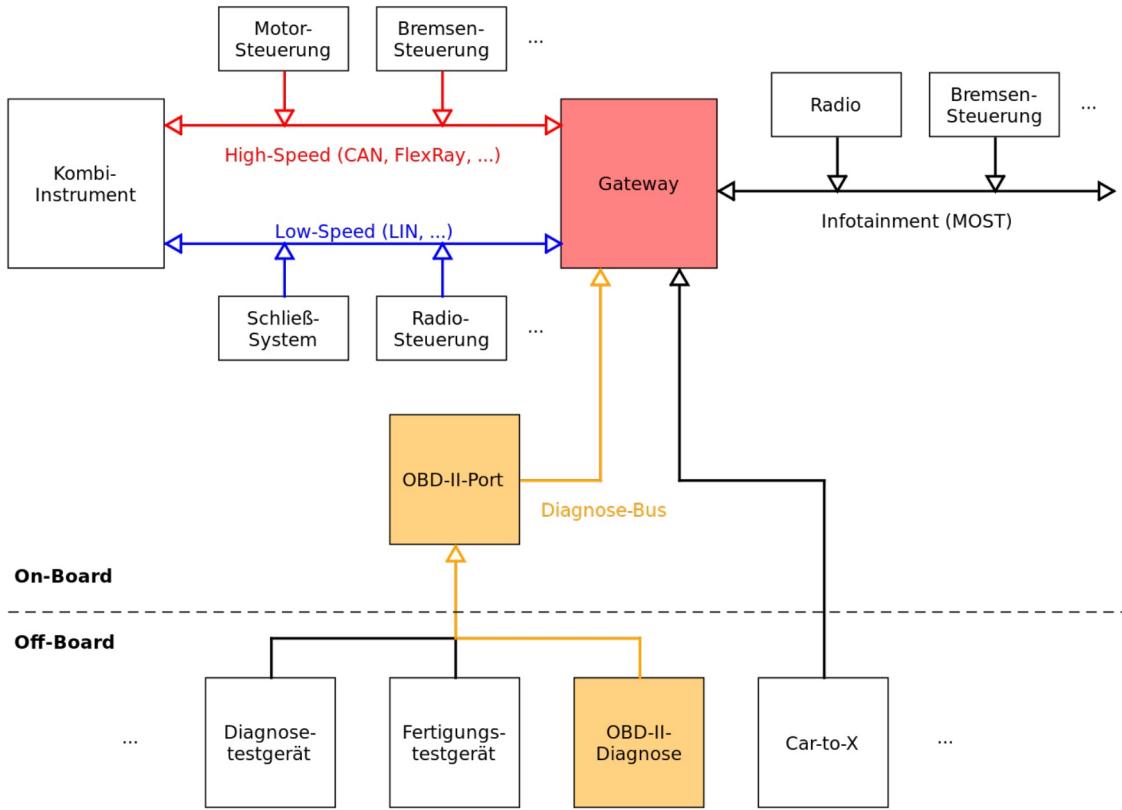


Abbildung 2: Fahrzeugnetzwerke: Typisches Bussystem moderner Fahrzeuge (stark vereinfacht)

Sensoren, Aktoren und Steuergeräte (engl. **Electrical Control Unit (ECU)**) kommunizieren untereinander mithilfe von Busnetzwerken. Aktoren setzen in Fahrzeugen Signale von Steuergeräten in mechanische Bewegungen um. Steuergeräte sind eingebettete Geräte, welche unter Zuhilfenahme von Sensordaten Abläufe regeln. Es besteht die Möglichkeit, dass Steuergeräte periodisch Ergebnis- oder Sensordaten als Broadcast versenden. Dies muss jedoch nicht der Fall sein – Steuergeräte können auch erst aktiv werden, sofern andere Steuergeräte dies explizit anfordern.

In Fahrzeugnetzen existieren häufig Gateways, welche als zentrale Knoten mehrere Bussysteme miteinander verbinden. Sie agieren somit als vermittelnde Stelle zwischen verschiedenen Bussystemen mit unterschiedlichen Geschwindigkeiten und Protokollen. Darüber hinaus können Gateways im Rahmen von Diagnosen abgefragt werden. Durch das Gateway wird zudem der Diagnosebus von den restlichen Bussen abgeschottet.

*Hinweis: Der Einsatz von zentralen Gateways widerspricht gewissermaßen der Busdefinition (siehe 2.1). Aus den oben genannten Gründen stellen zentrale Netzwerknoten jedoch essenzielle Komponenten von Fahrzeugbussen dar.*

### 2.2.1 On-Board-Kommunikation

Diese Kategorie umfasst die Kommunikation an Bord des Fahrzeugs und somit den Datenaustausch zwischen Aktoren, Sensoren und Steuergeräten des Automobils. Hierzu sind

diverse Bussysteme vorhanden:

- **High-Speed-Bus bzw. Echtzeitbus**

Sofern Daten in Echtzeit bzw. schnellstmöglich übertragen werden sollen, wird ein High-Speed-Bus eingesetzt. Über diesen werden besonders zeitkritische Daten übertragen. Dies sind zum Beispiel Motor-, Brems-, oder Getriebedaten. Eine verzögerte Übertragung könnte gravierende Folgen haben und beispielsweise das ABS beeinträchtigen. Charakteristisch ist die Übertragung zahlreicher Pakete, welche kleine Datenmengen transportieren. Ein Beispiel eines High-Speed-Busses ist der CAN-Bus.

- **Low-Speed-Bus**

Diese Busart wird zur Vereinfachung des Kabelbaums eingesetzt, insofern keine zeitkritischen Daten übertragen werden müssen. Dies ist etwa bei Sensordaten der Türen und bei Schließvorgängen der Fall. Low-Speed-Busse können, wie im Falle des LIN-Bus, kostengünstiger als CAN implementiert werden.

- **Infotainment-Systeme**

Multimediadaten besitzen ein hohes Datenvolumen, besonders zeitkritisch ist die Übertragung dieser jedoch nicht. Eine gewisse Latenzzeit soll aus Wiedergabegründen dennoch nicht überschritten werden. Aus diesem Grund existieren separate Busse für Infotainment-Systeme, wie zum Beispiel der **Media Oriented Systems Transport (MOST)-Bus**.

- **Fahrerassistenz-Systeme**

Systeme dieser Art erfordern oft einen Verbund der Anforderungen von High-Speed-Bussen und Infotainment-Systemen. So erfordern automatische Parkvorgänge, welche vom Fahrer überwacht werden, Echtzeitdaten die über den Infotainment-Bus ohne hohe Latenzen auf einem Bildschirm angezeigt werden.

## 2.2.2 Off-Board-Kommunikation

Sofern ein externes System mit einem Fahrzeug kommuniziert, findet eine Off-Board-Kommunikation statt. Dies ist unter anderem der Fall, wenn das Fahrzeug in einer Werkstatt einer Diagnose unterzogen wird. Dabei wird die standardisierte **On Board Diagnose (OBD)-II-Schnittstelle** verwendet, welche im Abschnitt 2.5.1 näher erläutert wird. Die bereits erwähnten Gateways sind oft mit einem Filtermechanismus ausgestattet, welcher Zugriffe seitens des OBD-II-Ports oder anderen Off-Board-Schnittstellen beschränken kann und nur für bestimmte Zwecke verwendete Daten akzeptiert. Detailinformationen bezüglich Filterung und Buszugriff folgen im Abschnitt 2.5.

Das Testgerät und das Fahrzeug einigen sich im Rahmen einer Diagnose auf ein Diagnoseprotokoll. Die meisten KFZ-Hersteller verwenden an dieser Stelle einen eigenen Standard [Wer14]. Aus diesem Grund unterstützen die Steuergeräte oft mehrere Standards. Es existieren jedoch Ansätze, welche eine herstellerübergreifende Lösung als Ziel haben (2.4.1).

Die Anforderungen an Diagnosesysteme reichen von der Umprogrammierung der Steuergerätekonfiguration, über das Testen der Funktionsfähigkeit verschiedener Komponenten

bis hin zur Aktualisierung der Steuergeräte-Firmware. Dabei muss stets die Fehlererkennung bezüglich der Übertragung, der Zugriffsschutz und die Autorisierung des Firmware-Updates berücksichtigt werden, um die Integrität der Systeme und deren Software zu gewährleisten.

Diese Aspekte werden meist als proprietäre Protokolle von Fahrzeugherstellern entwickelt und implementiert. Proprietäre Erweiterungen oder Modifikationen bestehender Standardmechanismen sind darüber hinaus zusätzlich möglich.

Zusätzlich existiert die Möglichkeit der Car-to-Car- bzw. Car-to-Infrastructure-Kommunikation [Wer14]. Allgemein wird dieser Vorgang als Car-to-X bezeichnet. Dabei erfolgt ein Datenaustausch mit anderen kommunizierenden Fahrzeugen oder externer Infrastruktur wie zum Beispiel automatischen Mautsystemen.

## 2.3 Übersicht der Busprotokolle

Das Grundwissen über zahlreiche Busprotokolle, deren Paketformate und deren grundlegende Arbeitsweise ist für die Analyse von Fahrzeugnetzwerken elementar. Ohne dieses Wissen kann keine ganzheitliche Analyse auf Netzwerk- oder Paketbasis durchgeführt werden. Aus diesem Grund folgt in diesem Abschnitt eine Einführung in gängige Fahrzeugprotokolle.

Eine generelle Unterteilung der Protokolle in Kategorien kann anhand des **Open System Interconnection (OSI)**-Modells vorgenommen werden [emoa]:

Tabelle 1: Fahrzeugprotokolle: OSI-Modell im Bezug auf Fahrzeugnetzwerke

Schicht	Schicht	Anwendung	Kategorie
7	Anwendung	Dienste bereitstellen (z.B. Fehlerspeicher auslesen)	Diagnose-Protokolle
6	Darstellung	Anzeige der Daten für den Benutzer	
5	Sitzungssteuerung	Authentifizierung, Synchronisation	
4	Transport	Segmentierung von Paketen, Stauvermeidung, Fehlerbehebung	Transport-Protokolle
3	Vermittlung	Routing, Adressierung, Fragmentierung	
2	Sicherung	Flusskontrolle, Fehlererkennung, Aufteilung von Nachrichten in Frames, Prüfsummen	Bus-Systeme
1	Bit-Übertragung	Signalkodierung, Signalpegel, (De)-Aktivierung von Verbindungen	

Die grau hinterlegten Schichten werden derzeit in Fahrzeugsystemen nicht *explizit* implementiert. Benötigte Aspekte der ausgelassenen Schichten werden jedoch bei Bedarf von anderen Schichten übernommen und implementiert.

Die unterstützte Geschwindigkeit von Fahrzeugprotokollen wird in sechs Klassen eingeteilt [Wer14]:

Tabelle 2: Fahrzeugprotokolle: Geschwindigkeitsklassen

Klasse	Bitrate
Diagnose	< 10Kbit/s
A	< 25Kbit/s
B	$25 \text{ Kbit/s} \leq x \leq 125\text{Kbit/s}$
C	$125 \text{ Kbit/s} \leq x \leq 1000\text{Kbit/s}$
D	> 1Mbit/s
Infotainment	> 10Mbit/s

Zur einfacheren Orientierung zwischen den zahlreichen verfügbaren Protokollen gibt folgende Tabelle eine Übersicht der wichtigsten Protokolle und den entsprechenden Schichten im obig vorgestellten OSI-Modell:

Tabelle 3: Fahrzeugprotokolle: Protokollübersicht

Schicht	Bus/Protokoll	Standard	Merkmale	Anwendung	Klasse
1-2 (Bus)	K-Leitung	ISO 9141	1 Diagnoseleitung Pin 7, optional Pin 15	Diagnose	A
	LIN	SAE J2602 ISO 17987-1	1 Diagnoseleitung	On-Board-Kommunikation Diagnose	A
	CAN CAN-FD TTCAN	ISO 11898 ISO 11992 ISO 11783	Verdrilltes Kabelpaar (CAN-High und CAN-Low) 250 oder 500 kBaud/s Terminierung durch Endwiderstände Pin 6 und 14	Diagnose On-Board-Kommunikation	B (Low-Speed) C (High-Speed)
	FlexRay	ISO 17458	Zeitscheibenverfahren	On-Board-Kommunikation	D
	MOST	Durch Herstellerkonsort. spezifiziert	Lichtwellenleiter Modi: MOST{25, 50, 150}	Infotainment	Infotainment
	Ethernet	IEEE 802.3 ISO 13400		Diagnose Firmware-Updates	10/100 Mbit
	ISO-TP	ISO 15765-2	Erweitert CAN, sodass längere Pakete möglich sind (= Segmentierung)	CAN	
4 (Transport)	FlexRay-TP	ISO 10681-2	FlexRay	FlexRay	
	TP 1.6 TP 2.0	Hausstandard	Verwendet durch VW-Konzern: VW, Audi, Skoda, Seat, ...	CAN	
	DoIP	ISO 13400-2		Ethernet	
	KWP2000	ISO 14230	K-Leitung, CAN Diagnose	OBD Diagnose	
7 (Anwendung)	UDS	ISO 14229 (UDS) ISO 15765 (UDS on CAN)	Herstellerübergreifende und standardisierte Diagnose, Methoden für direkten Speicherzugriff	OBD Diagnose	
	OBD	ISO 15031 US OBD, EU OBD	OBD-II-Port ist gesetzlich vorgeschrieben Standardisierte Schnittstelle	Diagnose	

Hinweis: Protokolle, welche hauptsächlich auf dem US-Markt vorzufinden sind wurden aus Gründen der Übersicht nicht in der vorherigen Tabelle dargestellt. Bei Bedarf können hierzu benötigte Informationen für die Protokolle PWM und VPW aus [Wer14] entnommen werden.

Fahrzeugprotokolle dienen meist einem fest definierten Zweck und können aufeinander aufbauen. Sie erweitern Funktionalitäten aus unteren Schichten bis hin zur Anwendungsschicht, welche die Interaktion mit den Fahrzeuggassen und deren Netzwerknoten erlaubt.

Ein beispielhafter Diagnosevorgang, welcher das Zusammenspiel der Schichten demonstriert, ist in der folgenden Abbildung zu sehen (angelehnt an [emoa] F. 6):

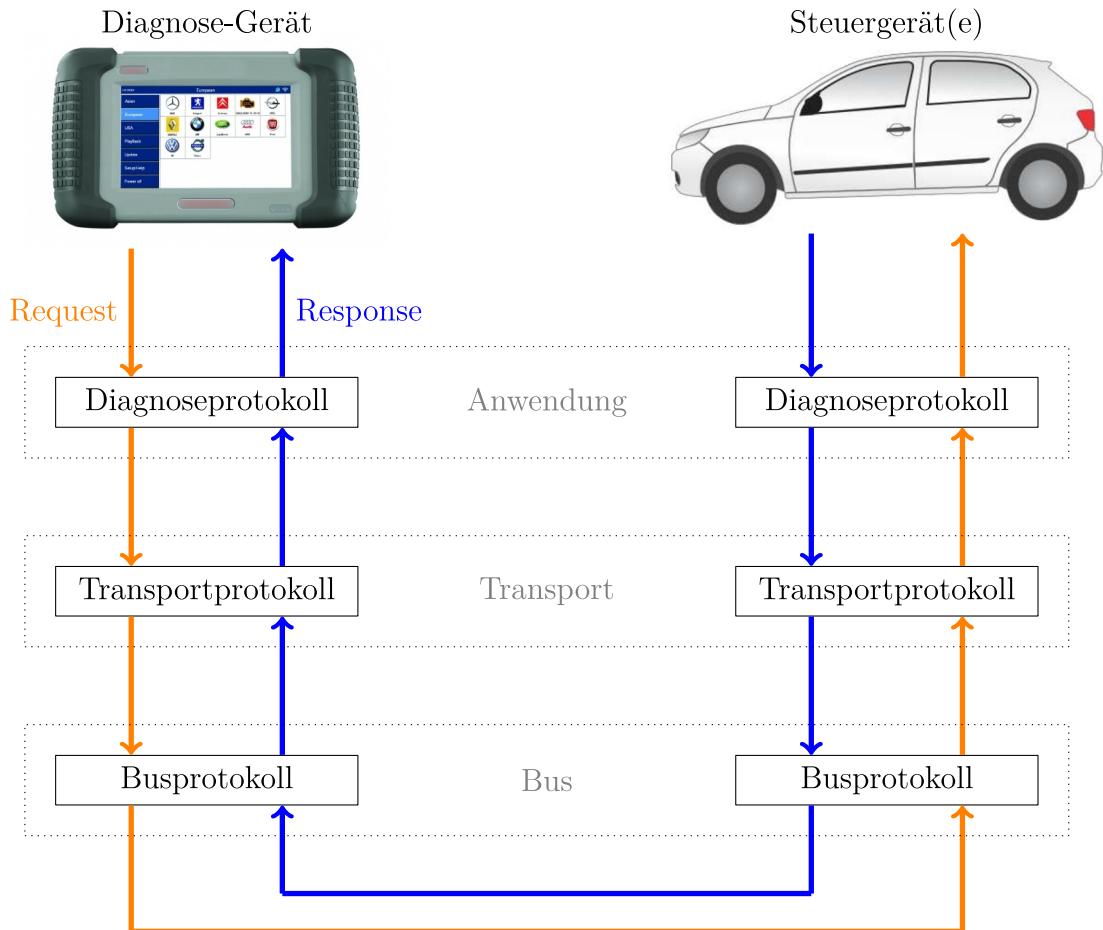


Abbildung 3: Fahrzeugprotokolle: Diagnosevorgang

Grafiken aus: [asvg], [diag].

Anfrage (Request) und Antwort (Response) durchlaufen somit gemäß dem OSI-Modell die oberen Schichten in Richtung der Busschicht und wieder zurück über die Transportschicht zu der Anwendungsschicht. Hierbei sind die einzelnen Schichten *theoretisch* austauschbar. In der Praxis liegen jedoch oft Abhängigkeiten zwischen einzelnen Schichten vor.

Mittels folgender Grafik [emh] können die verfügbaren Fahrzeugprotokolle grob zeitlich, technisch und finanziell eingeordnet werden:

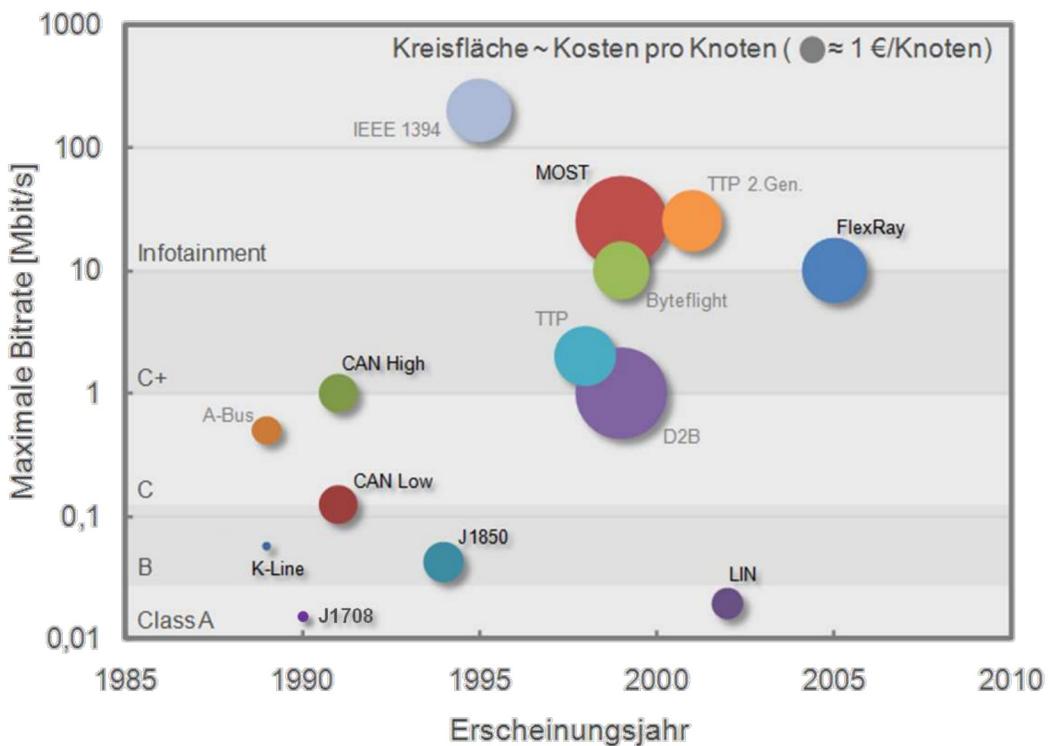


Abbildung 4: Fahrzeugprotokolle: Kosten- und Erscheinungsübersicht

Es ist erkennbar, dass eine Relation zwischen der maximalen Bitrate und den Implementierungskosten besteht. Die Kosten in den Bereichen der Mikrocontroller und den Netzwerkbausteinen verringern sich zwar ständig, jedoch sind die Kosten vor allem im Infotainment- und Hochleistungsbereich nach wie vor am höchsten. Der Trend des stetigen Netzwerkausbau im Automobilbereich ist anhand der entwickelten Protokolle nachzuvollziehen. Mit der Ausnahme von LIN erschienen circa ab dem Jahr 1995 fast ausschließlich Busprotokolle mit hohen maximalen Übertragungsgeschwindigkeiten.

Nachfolgende Abschnitte dienen als Einführung in die verschiedenen Protokolle und deren Zusammenspiel. Der Umfang eines jeweiligen Abschnitts spiegelt dabei nicht die Komplexität und den Funktionsumfang eines Protokolls wider, sondern die technische Relevanz bei Fahrzeuganalysen gängiger Automobile im Rahmen dieser Arbeit.

*Alle Informationen zu nachfolgenden Protokollen stammen mit Ausnahme von Tabellen und Abbildungen, sofern nicht abweichend angegeben, aus [Wer14].*

### 2.3.1 K-Leitung / K-Line mit KWP2000

Als ältestes Fahrzeugprotokoll, welches in europäischen Fahrzeugen vorgefunden werden kann, ist die K-Leitung bzw. K-Line zu nennen. Es wurde in den 1980er-Jahren in Form eines Industriestandards entwickelt und 1989 als **ISO 9141** spezifiziert. Diese Spezifizierung umfasst die Schichten 1-2 (Bit-Übertragungs- und Sicherungsschicht) – eine gesonderte Transportschicht wird an dieser Stelle *nicht* definiert. K-Line dient unter anderem als Basis für das Anwendungsprotokoll KWP2000 [**Smi16**] (**ISO 14230**), dessen Bus-Merkmale

aus Verständnisgründen bereits innerhalb dieses Abschnitts aufgegriffen werden. Gesichtspunkte von KWP2000 auf Applikationsebene werden im Zusammenhang mit **Unified Diagnostic Services (UDS)** in einem folgenden Abschnitt (2.4) beschrieben. KWP2000 baut auf K-Line auf und wird oft mit diesem Protokoll in Kombination verwendet.

Daten werden bei K-Line über eine einzelne Leitung ausgetauscht – die K-Leitung. Die dabei aufgebaute Verbindung ist bidirektional. Optional ist zusätzlich der Einsatz einer zweiten Leitung, der L-Leitung, möglich. Diese wird laut Standard jedoch nur für die einmalige Initialisierung der Verbindung benötigt. *Theoretisch* ist es über die L-Leitung möglich proprietäre Funktionalitäten umzusetzen. Die gemeinsame Verwendung von Geräten mit und ohne L-Leitung innerhalb eines Fahrzeugs ist laut Standard erlaubt.

Theoretisch ist mit K-Line ein Nachrichtenaustausch der Steuergeräte innerhalb des Fahrzeugs möglich. In der Praxis wird K-Line meist jedoch nur zur Diagnose verwendet.

Diese Verbindung umfasst ein Testgerät des Herstellers und ein einzelnes Steuergerät des Fahrzeugs. Dabei ist das Testgerät der Master und das zu diagnostizierende Steuergerät der Slave innerhalb der Verbindung. Dies bedeutet, dass der Slave eine passive Rolle einnimmt und lediglich auf Anfragen des Masters antwortet.

Mittels K-Line können in Verbindung mit KWP2000 **OBD-II**-Anfragen im Rahmen von Diagnosesitzungen an kompatible Fahrzeuge versandt werden.

Eine Diagnosesitzung, welche beispielsweise **OBD-II** als Anwendungsschicht verwendet, wird in drei Phasen abgewickelt:

1. Initialisierung (Verbindungsaufbau)
2. Datenaustausch
3. Verbindungsabbau

### **Initialisierung:**

Durch den Austausch von sogenannten Keywords einigen sich Steuergerät und Tester auf ein Diagnoseprotokoll. Zusätzlich stellt sich das Testgerät auf die Baudate im Bereich von 1,2kBd bis 10,4kBd ein. Baud ist die für Bussysteme typische Einheit der Übertragungsgeschwindigkeit. Ein Baud entspricht der Anzahl der übertragenen Symbole pro Sekunde. Der Schritt der Geschwindigkeitssynchronisation ist notwendig, da sonst keine intakten Datenpakete empfangen werden können.

Die ausgetauschten Keywords und darauf aufbauende Diagnoseprotokolle waren für lange Zeit nicht spezifiziert. Selbst für entsprechende Steckverbindungen lag für lange Zeit kein herstellerübergreifender Standard vor. All diese Aspekte wurden proprietär implementiert und selbst nach einer Erweiterung der Spezifikation (ISO 9141-2) existieren heutzutage noch proprietäre Keywords, welche über die gesetzlich vorgeschriebenen emissionsrelevanten Funktionen des erweiterten Standards hinaus gehen. Diese vom Gesetzgeber geforderten Funktionen dienen zur Diagnose von eventuellen Problemen im Bereich Schadstoffemission.

### **Datenaustausch:**

Nachrichten bzw. Anfragen werden, wie bereits beschrieben, mittels Request-Response abgewickelt. Dabei wird folgendes Paketformat verwendet:

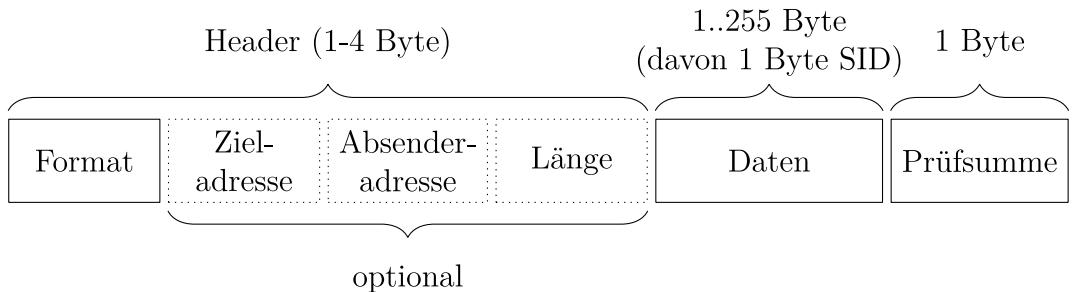


Abbildung 5: Fahrzeugprotokolle: KWP2000: Paketformat auf Busschicht

Mithilfe des Format-Flags wird angegeben, ob die optionalen Ziel- und Absenderadressen gesetzt sind. Der Wert des Längenfeldes gibt die Anzahl der Bytes im Datensegment an. Hierfür existiert ein Spezialfall: Sofern weniger als 64 Bytes übertragen werden sollen, kann die Länge des Datenfeldes auch über das Format-Flag kodiert werden. Dies bewirkt, dass das Längenfeld nicht mehr verwendet und daher nicht versendet werden muss.

Bezüglich der Adressen gibt es zwei Fälle. Beide Adressarten umfassen jeweils 8 Bit. Physische Adressen werden von den entsprechenden Fahrzeugherstellern selbst für individuelle Steuergeräte innerhalb des Fahrzeugs vergeben. Diese Adressen sind *nicht* öffentlich verfügbar und müssen mittels Analysen und Reverse-Engineering bestimmt werden, um weitergehende Untersuchungen zu ermöglichen. Zum anderen existieren funktionale Adressen. Für gesetzlich vorgeschriebene emissionsrelevante Normen sieht der Gesetzgeber den Zwang einer festgelegten und stets bekannten Adresse vor. Aus diesem Grund werden funktionale Adressen implementiert, mittels welcher Anfragen ohne Kenntnis der konkreten physikalischen Adresse gestartet werden können.

Keywords, welche in der Initialisierungsphase benutzt werden, verwenden obige Header-Datenstrukturen zur Übertragung der verwendeten Header-Optionen an Diagnosegeräte. Darüber hinaus werden hierdurch Timings bezüglich der Übertragung festgelegt.

Im Datensegment bildet das erste Byte den **Service Identifier (SID)**. Der Wert des SID-Bytes legt fest, wie die restlichen Bytes des Datenfeldes interpretiert und verarbeitet werden. Ein einzelner Service bietet im Standard spezifizierte Diagnosemethoden, proprietären Zugriff oder sonstige Interaktionsmöglichkeiten an. Mittels Reverse-Engineering, Datenverkehr-Analyse des Busses, Fuzzing oder weiteren entsprechenden Methoden können auch proprietäre SIDs ermittelt werden, um den Fahrzeugbus weitergehend zu analysieren.

Die Prüfsumme berechnet sich aus der Summe aller Byte-Werte modulo 256.

Neben dem häufig verwendeten KWP2000-Paketformat definiert K-Line mit ISO 9141-2 ein eigenes Paketformat [[ISO9141-2](#)], welches jedoch aufgrund der Existenz von KWP2000-Paketen selten verwendet wird:

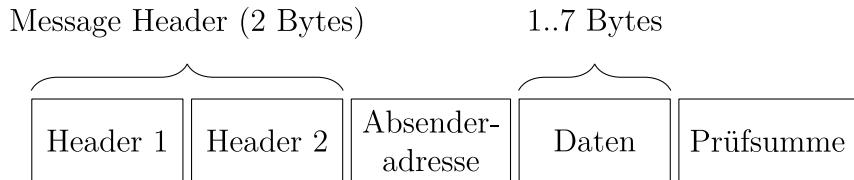


Abbildung 6: Fahrzeugprotokolle: K-Line: Paketformat (nach ISO 9141-2)

Das Header-Segment schreibt keine konkrete Bedeutung der beiden enthaltenen Bytes dar. Einigen Quellen kann jedoch entnommen werden, dass in der Praxis für die beiden Header-Bytes die Bedeutung *Priorität* und *Zieladresse* getroffen wird [Smi16]. Die Prüfsumme des beschriebenen Paketformats ist laut Spezifikation identisch zu der Prüfsumme von KWP2000-Paketen. Eine Verwendung des **Cyclic Redundancy Check (CRC)**-Prüfsummenverfahrens an dieser Stelle scheint in der Praxis jedoch nicht ausgeschlossen zu sein [Smi16].

Das KWP2000-Protokoll kann anstelle von K-Line mit dem **CAN**-Busprotokoll verwendet werden [emoa]. Dieser Vorgang wird als *KWP2000 on CAN* bezeichnet.

Bevor KWP2000 spezifiziert wurde, implementierten viele Automobilhersteller ein eigenes proprietäres Protokoll auf der Basis von K-Line. Beispiele hierzu sind KW 71 (BMW), KW 81/82 (Opel) und KW1281 (VW/Audi). In seltenen Fällen sind diese Protokolle noch heute in älteren Fahrzeugmodellen vorzufinden.

KWP2000 wird heutzutage vermehrt von **UDS** verdrängt. Selbiges gilt für K-Line, welches durch **CAN** ersetzt wird. Heutzutage existieren jedoch noch zahlreiche Fahrzeuge mit K-Line und KWP2000, weshalb beide Protokolle im Rahmen dieses Abschnitts aufgearbeitet wurden.

### 2.3.2 CAN

Die Spezifikation **ISO 11989** [crb] beschreibt das CAN-Protokoll (Controller Area Network) für PKWs, welches von der Robert Bosch GmbH seit Mitte der 1980er-Jahre entwickelt wird. CAN ist für Nutzfahrzeuge (ISO 11783) und LKWs (ISO 11992) spezifiziert und definiert beide Busschichten (Bit-Übertragungs- und Sicherungsschicht). Es ist eines der aktuell am häufigsten verwendeten Protokolle im modernen Fahrzeugen.

Über ein verdrilltes Kabelpaar erfolgt die Datenübertragung im CAN-Bus. Die Leitungen werden als CAN-Low und CAN-High bezeichnete. Low und High haben hierbei jedoch *keinen* Zusammenhang mit der eigentlichen Datenrate des jeweiligen Busses. Vielmehr hängen diese Bezeichnungen mit der Übertragungsart des Busses zusammen.

Auf dem CAN-Bus liegt im Falle von High-Speed-CAN auf beiden Leitungen eine konstante Spannung von 2,5V an. Erfolgt die Übertragung eines 0-Bits, wird auf der CAN-High-Leitung die Spannung um 1V reduziert und auf der CAN-Low-Leitung entsprechend um 1V erhöht. Im Falle von Low-Speed-CAN liegt der Ruhepegel auf 5V für CAN-Low bzw. 0V für CAN-High. Wird ein 0-Bit übertragen, wird die Spannung von CAN-High auf ~3,6V angehoben und jene von CAN-Low auf ~1,4V verringert. Diese Übertragungsart wird als differentielle Signalübertragung bezeichnet und soll Übertragungsstörungen

vermeiden [isg]. Darüber hinaus wird hierdurch die Identifikation von CAN-Leitungen ermöglicht (2.5.2), um Analysemaßnahmen vorzunehmen.

Die schematische Anbindung von Steuergeräten an CAN-High und CAN-Low ist folgender Abbildung zu entnehmen (angelehnt an [Wer14] S. 59):

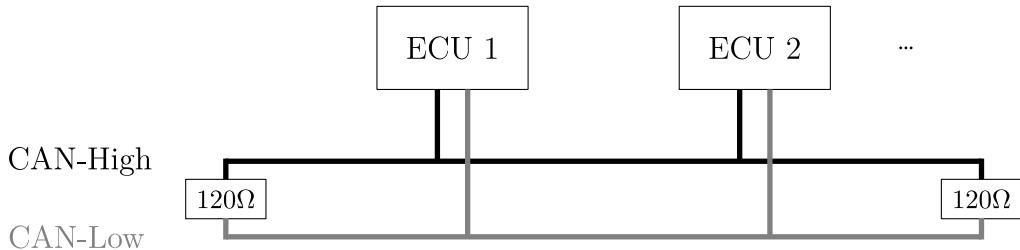


Abbildung 7: Fahrzeugprotokolle: CAN: Schematischer Aufbau (High-Speed-CAN)

Die maximale Datenrate beträgt bei CAN 1Mbit/s – ein Vielfaches der Übertragungsgeschwindigkeit von K-Line. Es werden jedoch oft geringere Datenraten verwendet, wie an folgenden Kategorien der CAN-Spezifikation zu erkennen:

- **High-Speed-CAN:** Mit ISO 11989-2 ist ein CAN-Bus mit hohen Datenraten spezifiziert. High-Speed bedeutet in diesem Bereich  $\geq 250\text{Kbit/s}$ . Gängige Datenraten im PKW-Bereich betragen 500Kbit/s. Einsatzgebiete umfassen zum Beispiel zeitkritische Anwendungen wie die Übertragung von Motor- oder Bremsdaten. Ein Abschlusswiderstand von  $120\Omega$  verhindert Reflexionen am Ende der Busleitungen. Dieser Umstand ermöglicht erneut die Identifikation von CAN-Bus-Leitungen (2.5.2). Bei der Busanalyse ist im Falle von Modifikationen des Busses stets auf die Erhaltung der Terminierungswiderstände zu achten [Smi16].
- **Low-Speed-CAN:** CAN-Busnetze mit niedrigeren Datenraten mit bis zu 125Kbit/s werden durch ISO 11989-3 als Low-Speed-CAN definiert. Durch die niedrigere Datenrate ergibt sich der Vorteil, dass die Datenleitungen entsprechend über längere Distanzen hinweg störungsfrei verlegt werden können. Ein Abschlusswiderstand wird nicht benötigt. Der Einsatz erfolgt beispielsweise bei der Karosserieelektronik. Gängige PKW-Datenraten im Low-Speed-Bereich betragen zwischen 100Kbit/s und 125Kbit/s.

Vereinzelt ist auch die Anwendung einer Zwischenstufe beider obig beschriebenen Geschwindigkeiten aufzufinden. Diese Netzwerke werden als Mid-Speed-CAN bezeichnet.

Die Weiterleitung von Nachrichten erfolgt gemäß einer **Vehicle Message Matrix (VMM)** oder K-Matrix, wobei der Buchstabe *K* für Kommunikation steht. Diese Matrix wird seitens der Automobilhersteller *nicht* veröffentlicht und ist somit für die Öffentlichkeit geheim. Lizenzkosten zur Einsicht und Nutzung *einiger* proprietärer Informationen in diesem Bereich belaufen sich zum Beispiel für BMW auf 25.000\$ – mit einer jährlichen Aktualisierungsgebühr von 2000\$ [eti]. Eine Einsicht ist somit nicht für Jedermann gegeben.

In der K-Matrix wird unter anderem folgendes festgelegt [hjs]:

- Zuordnung zwischen Sender -und Empfängersteuergerät, Nachricht und deren Struktur, Nachrichtenbedingung und Zykluszeit
- Aufbau und Größe der Nachricht
- Nachrichten-ID

Die ID einer Nachricht legt neben Informationen zur Interpretierung des Pakets gleichzeitig auch die Priorität der Nachrichtenübertragung dieser Nachricht auf dem Bus fest. Je niedriger die ID, desto höher ist die Priorität des Pakets auf dem Bus.

Der schematische Aufbau von Standard-CAN-Frames ist auf folgender Abbildung [cfr] am Beispiel von High-Speed-CAN zu sehen:

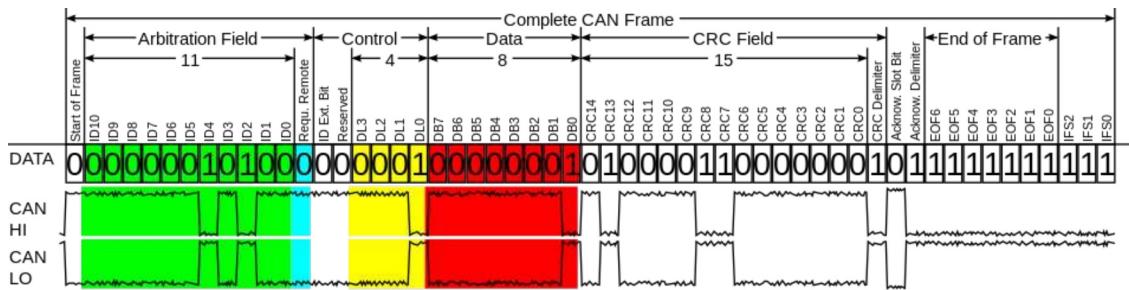


Abbildung 8: Fahrzeugprotokolle: CAN-Paketformat

Das **Remote Transmission Request (RTR)-Bit**, in der obigen Grafik als **Requ. Remote** bezeichnet, zeigt an, ob Daten eines bestimmten Typs von einem anderen Netzwerknoten angefordert werden. Alle nachfolgenden Bits innerhalb des **Control**-Segments beziehen sich auf Extended-Pakete.

Es folgt das Datensegment, welches bis zu 8 Bytes umfassen kann. Dies ist in obiger Grafik durch acht  $DB_x$  genannte Bytes dargestellt. Es besteht die Möglichkeit eines Padding-Zwangs [Smi16]. Dies bewirkt, dass alle verarbeiteten Pakete eine Nutzdatenlänge von acht Bytes besitzen müssen. Der Einsatz des Paddings ist jedoch systemabhängig.

Empfangsbestätigungen werden anhand des ACK-Bits (*Acknow. Slot Bit*) in Kombination mit einem entsprechenden Delimiter in Antwortpaketen angezeigt.

Die drei letzten Bits definieren einen Zwischenraum zwischen nachfolgenden CAN-Frames.

Das eingesetzte Übertragungsverfahren wird als inhaltsbezogene Adressierung bezeichnet. Dabei zeigt die CAN-ID die Art der übertragenen Daten an und *nicht* etwa eine Adresse des Paketempfängers. Auf diese Art und Weise entscheidet jeder CAN-Knoten individuell, ob eine erhaltenes Paket verarbeitet oder ignoriert wird. Versendet ein Steuergerät beispielsweise die aktuelle Motordrehzahl, muss dieses nicht Sorge tragen, dass jedes Steuergerät, welches derartige Informationen benötigt, kontaktiert wird.

Die aktuelle CAN-Spezifikation sieht zwei Arten von CAN-Paketen vor: CAN 2.0A und CAN 2.0B. Pakete vom Typ A stellen Standard-CAN-Frames dar, welche eine 11-Bit-ID (Arbitration ID) besitzen. Dies ist auf obiger Abbildung zu erkennen. Bei Paketen vom Typ B (Extended-Format) hat diese ID einen Umfang von 29 Bit.

Extended-Frames (CAN 2.0 B) stellen eine nachträgliche Protokollerweiterung dar. Die Aktualisierung des Standards erlaubt durch insgesamt längere CAN-IDs mehr Knoten bzw. durch IDs ansprechbare Services auf einem CAN-Bus. Das **Substitute Remote Request (SRR)**-Bit ersetzt bei Extended-Paketen das RTR-Bit. Im Falle eines Extended-Pakets folgen nach dem **Identifier Extension (IDE)**-Segment die restlichen Bits der ID, gefolgt von einem RTR-Bit. Es gibt öffentlich einsehbare IDs (siehe 2.5.1), aber auch proprietäre IDs von Fahrzeugherstellern als Teil der K-Matrix. Im Falle von Typ A ist zusätzlich noch ein Steuerbit im ID-Feld enthalten – für Typ B sind dies 3 Bits.

Die Kontrollbits umfassen [Smi16]:

1. IDE (Identifier Extension): Gibt an, ob nach den ursprünglichen 11 ID-Bits weitere 18 ID-Bits folgen, um ein CAN-Frame im Extended-Format zu definieren
2. Reserviert
3. Länge des Datenfeldes im Bereich {0..8}

Folgendes Schema zeigt die Bit-Sequenzen [mes] eines Standard-Frames im Vergleich zu einem Extended-Frame:

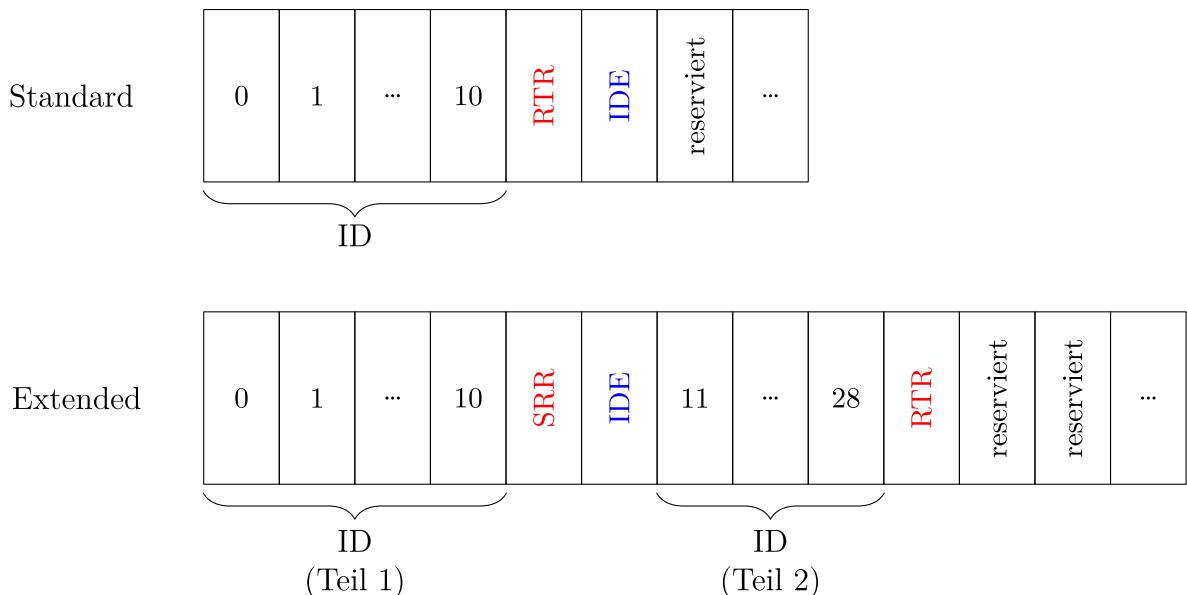


Abbildung 9: Fahrzeugprotokolle: CAN: Vergleich von Standard- und Extended-Frames

Es existiert eine Reihe von Transportprotokollen, welche auf dem CAN-Protokoll basieren und dieses erweitern. Unter anderem wird hierdurch die Limitierung von nur 8 Datenbytes pro CAN-Paket aufgehoben. Folgende Abschnitte beschreiben einige der am häufigsten in Fahrzeugen vorzufindenden CAN-Erweiterungen:

**ISO-TP (ISO 15765-2):** Durch dieses Transportprotokoll ist es unter anderem möglich, dass die Protokolle KWP2000 und UDS unter CAN verwendet werden können. Dabei werden die Absender- und Empfängeradressen von KWP2000 und UDS durch CAN-IDs

abgebildet. Neben der Flusssteuerung wird zusätzlich eine Paketsegmentierung definiert, sodass Datensätze mit bis zu 4095 Byte übertragen werden können.

Insgesamt werden durch ISO-TP vier Paketformate definiert. Drei davon können in folgender Übersicht eingesehen werden:

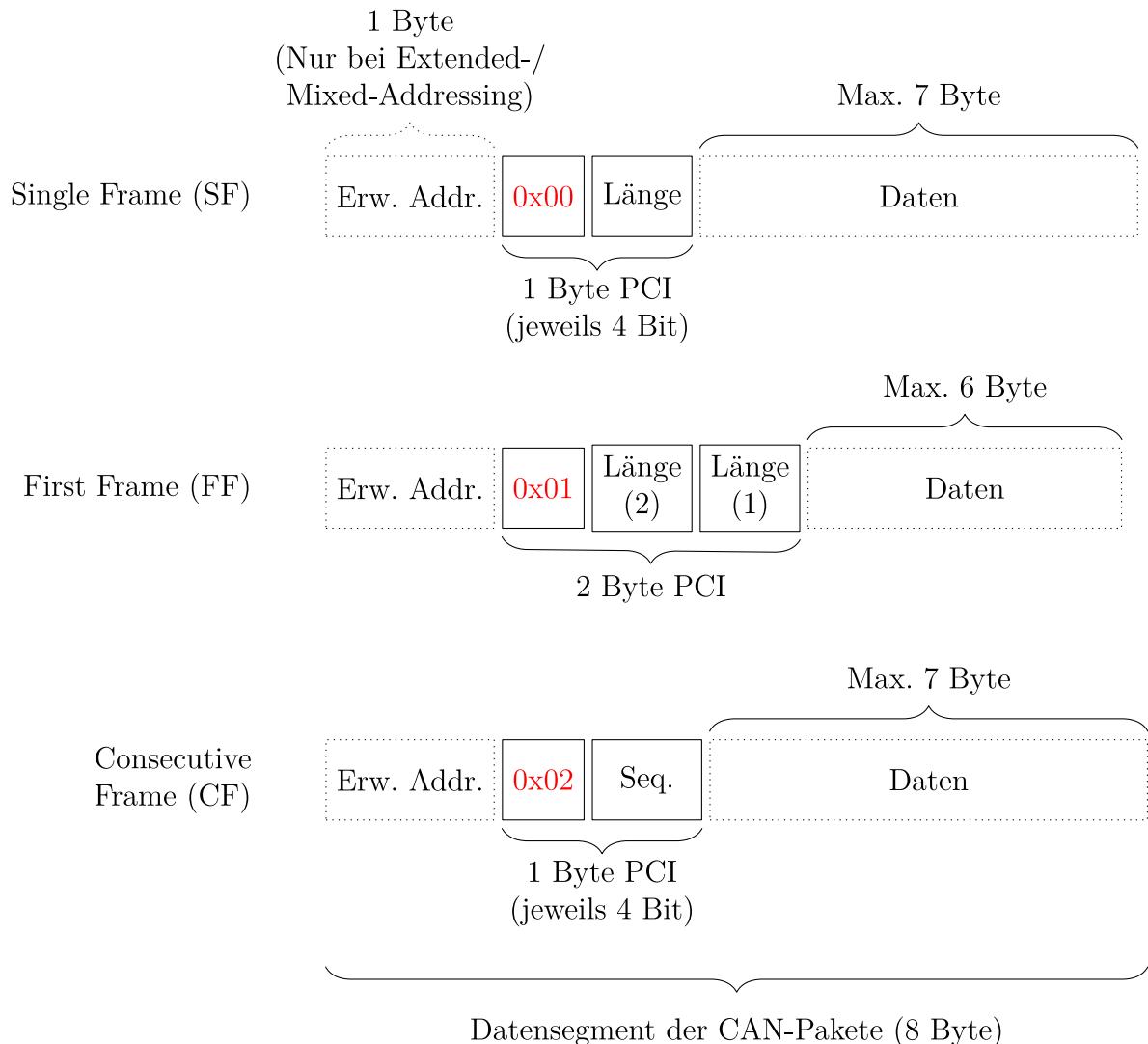


Abbildung 10: Fahrzeugprotokolle: ISO-TP: Paketformate

Da ISO-TP auf CAN aufbaut, werden alle obig definierten Pakete in den Nutzdaten von CAN-Paketen versandt.

Sofern im Fahrzeugnetzwerk Gateways eingesetzt werden, kommen Extended- und Mixed-Addressing zur Anwendung. Hierbei wird das erste Byte des ISO-TP-Pakets (AI-Byte) als erweiterte Adressinformation verwendet. Entsprechend verringert sich das Volumen des Datensegments hierbei um ein Byte. Das folgende **Protocol Control Information (PCI)**-Byte bzw. die folgenden PCI-Bytes enthalten Informationen über den Pakettyp und die Länge der Nutzdaten.

Bei der Datenübertragung treten zwei Fälle auf: Sofern der 7-Byte-Bereich für Nutzdaten des Single Frames für die zu übertragenden Daten ausreicht, wird ein solches Single

Frame versendet. Sollte dies nicht der Fall sein, wird mittels eines First Frames und mehreren Consecutive Frames die segmentierte Datenübertragung über mehrere Pakete hinweg durchgeführt. Im Falle eines First Frames sind zwei Längenfelder enthalten. Dabei geben beide Felder zusammen die Gesamtlänge der zu übertragenden Daten an – inklusive den noch in den Consecutive Frames zu übertragenden Daten. Das erste Längenfeld stellt dabei die höchstwertigsten Bits dar. Es werden solange Consecutive Frames versandt, bis alle Nutzdaten übertragen wurden. Das SN-Feld (Seq.) gibt bei derartigen Frames die aktuelle Sequenznummer an. Die enthaltenen Werte innerhalb des Sequenznummern-Feldes werden modulo 16 interpretiert.

Das vierte Paketformat von ISO-TP stellt das Flow Control Frame dar. Hierdurch gibt der Empfänger von Consecutive-Frame-Paketen Rückmeldung an den Absender, um eine eventuelle Anpassung der Datenrate zu erwirken.

ISO-TP definiert Broadcast-Adressen für 11-Bit- und 29-Bit-CAN. Diese werden beispielsweise bei der Erstellung von Diagnose-Anfragen verwendet. Die funktionale Adresse 0x7DF gilt für 11-Bit-Pakete, während die Adresse 0x18DB33F1 für 29-Bit-Pakete verwendet wird.

### TP 1.6/2.0:

Beide Protokolle der TP-Familie werden durch Industriestandards spezifiziert. TP 1.6 und dessen Nachfolger, TP 2.0, stammen aus dem Hause Volkswagen. Es liegt daher nahe, dass dieses Protokoll häufig bei Automobilen dieses Konzerns vorzufinden ist. Ein Teil der Spezifikation ist in Fachbüchern zu finden [Wer14], während die ungekürzte Spezifikation auf interne Dokumente des VW-Konzerns zurückgeht.

Mittel sogenannter Kanäle besitzt dieses Protokoll die Eigenschaft, dass eine Verbindungsorientierung vorliegt. Dabei wird für zwei Teilnehmer eine exklusive Verbindung bereitgestellt – ein Kanal. Jeder Kanal besitzt eine eigene reservierte Kanal-ID. Diese wird dynamisch zwischen beiden teilnehmenden Netzwerkknoten ausgehandelt.

TP 1.6/2.0 definieren diverse verschiedene Paketformate und verfügen über einen Mechanismus zur Flusskontrolle. Broadcast-Verbindungen und die Übertragung beliebig großer Datenblöcke sind zudem bei TP 2.0 möglich, jedoch nicht bei TP 1.6. Ob ein Steuergerät Broadcast-Nachrichten versendet ist Teil der herstellerseitigen Implementierung und daher nicht explizit Teil der Protokollspezifikation.

Broadcast-Nachrichten sind wie folgt aufgebaut:

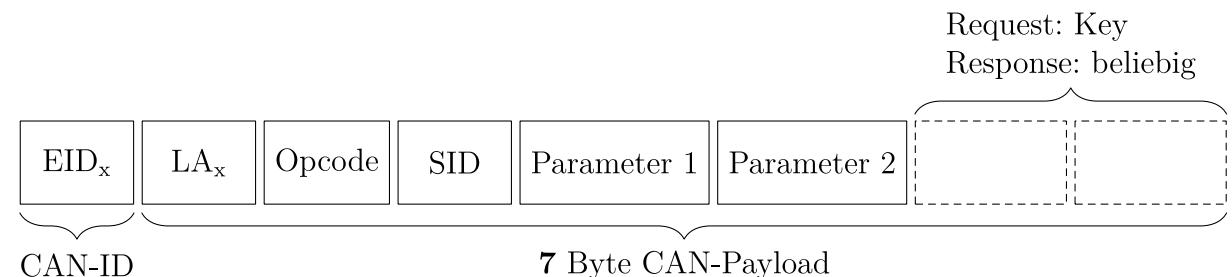


Abbildung 11: Fahrzeugprotokolle: TP 2.0: Broadcast-Nachrichten

Der Wert EID<sub>x</sub> stellt die Eröffnungs-ID des Netzwerkknotens X dar. Dieser Wert wird aus der Addition der Basis-Adresse des Netzwerks und der logischen Adresse (LA<sub>x</sub>)

des Netzwerknotens berechnet. Da TP1.6/2.0 CAN als Basisprotokoll verwendet, wird bei Broadcast-Nachrichten die Eröffnungs-ID anstelle der CAN-ID verwendet. Durch TP1.6/2.0 werden insgesamt sieben der acht Nutzdaten-Bytes, wie auf obiger Abbildung zu erkennen, verwendet.  $LA_x$  dient dem Routing der Pakete über Gateways und eventuellen Netzwerkgrenzen hinweg. Durch den Opcode wird der Nachrichtentyp definiert. Dieser besitzt den Wert 0x23, falls der Broadcast eine Anfrage darstellt. Andererseits ist der Wert 0x24 für Broadcast-Antwortpakete möglich.

Durch andere Steuergeräte oder Steuergerätegruppen angebotene Dienste und Funktionen können mittels SID (Service Identifier) angesprochen werden. Diese und weitere Dienste werden auf der Anwendungsschicht definiert. Parameter für kontaktierte Dienste werden in den beiden dazugehörigen Paketsegmenten übertragen.

Die letzten beiden Bytes dienen zur Markierung, dass von einem Paketempfänger eine Antwort erwartet wird. In diesem Fall enthalten beide Bytes, welche den sogenannten Key definieren, den Wert 0x00. Bei einer Broadcast-Antwort dürfen an dieser Stelle beliebige Daten eingetragen werden. Dies erlaubt in der Praxis zusätzlichen Raum bei der Implementierung von herstellerspezifischen Protokollen und Paketen.

Wesentlich einfacher sind im Gegensatz Pakete aufgebaut, die über bestehende Kanäle ausgetauscht werden:

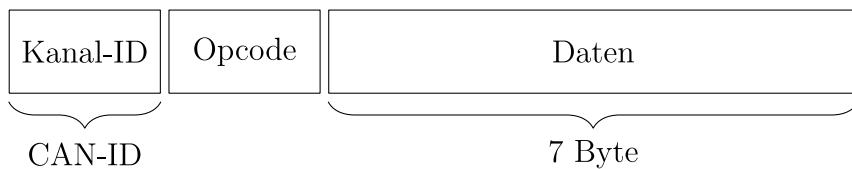


Abbildung 12: Fahrzeugprotokolle: TP1.6/2.0: Paketaufbau

Der Opcode enthält diverse Steuerungsbits, um den korrekten und effizienten Ablauf der Verbindung zu gewährleisten. Enthalten sind unter anderem Sequenznummern und Flags bezüglich der Endemarkierung einer Nachricht oder der Empfangsbestätigung. Jedes derartige Datenpaket wird mit einem ACK-Paket vom Empfänger bestätigt.

Neben dem Fehlen von Broadcast-Paketen unterscheidet sich TP 1.6 von TP 2.0 unter anderem darin, dass keine Paketformate zum Verbindungstest und der Unterbrechung von Übertragungen verfügbar sind.

### CAN-FD:

Als weitere CAN-Erweiterung wurde CAN-FD im Jahr 2012 von der Robert Bosch GmbH vorgestellt [fdvp] und unter ISO 11898-7 [fdvs] spezifiziert. FD steht in diesem Kontext für Flexible Data Rate. Ziele sind die Effizienzsteigerung im Vergleich zu traditionellem CAN-Traffic und die Erhöhung des Datendurchsatzes [fdvb]. Die Geschwindigkeit ist zwischen High-Speed-CAN (1 Mbit/s) und dem FlexRay-Protokoll (10Mbit/s) angesiedelt. Je nach Anforderung und verwendeter Hardware sind Datenraten von 2-4 Mbit/s innerhalb der Fahrzeugnetzwerke und Datenraten von 8 Mbit/s für Software-Aktualisierungen in der Praxis möglich [fdvp]. Dabei ist die Erhöhung der Bitrate unabhängig von der Leitungslänge [fdvs]. Die Implementierungskosten pro Netzwerknoten sind nur geringfügig höher als jene von herkömmlichen CAN-Netzen.

Es werden innerhalb des Control-Segments von CAN-Paketen (siehe Abbildung 8) drei neue Bits definiert:

1. **Extended Data Length (EDL)**: Unterscheidung von CAN- und CAN-FD-Frames, beansprucht das erste reservierte Bit des Control-Segements von Standard-CAN
2. **Bit Rate Switch (BRS)**: Erhöhung der Bitrate während der Übertragung des Daten-Segments
3. **Error State Indicator (ESI)**: Zeigt an, ob sich ein CAN-FD-Knoten im Fehlerzustand befindet

CAN-FD-Pakete haben eine maximale Nutzdatenlänge von 64 Bytes, was eine deutliche Vergrößerung im Vergleich zu Standard-CAN darstellt.

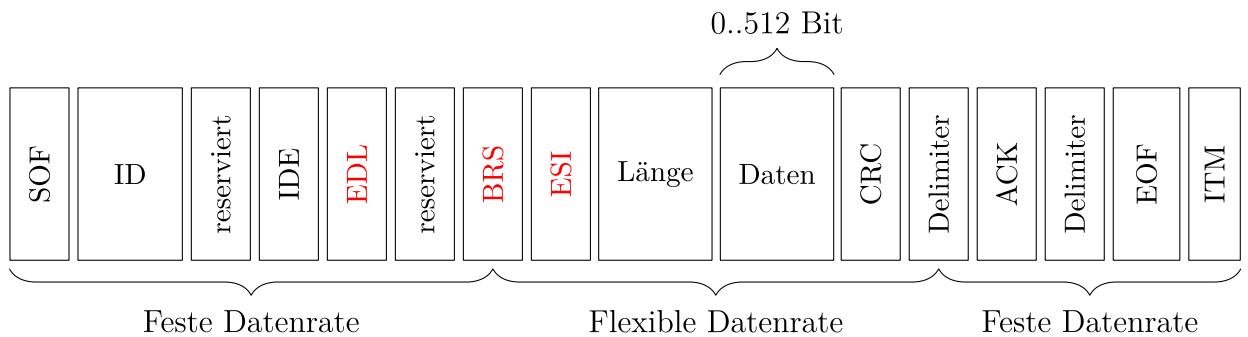


Abbildung 13: Fahrzeugprotokolle: CAN-FD: Paketaufbau

Das ID-Feld ist jeweils für 11-Bit- und 29-Bit-IDs ausgelegt [fdbp]. Das Volumen der Nutzdaten wurde auf 512 Bits bzw. 64 Bytes vergrößert. Entsprechend erfolgt eine Anpassung des CRC-Generator-Polynoms, um die erweiterten Nutzdatenfelder zu verarbeiten [fdbp]. Für Standard-Frames (11 Bits) werden 17 CRC-Bits verwendet, während für erweiterte Frames 21 Bits als CRC-Prüfsumme verwendet werden.

Neben dem Paketaufbau ist zusätzlich erkennbar, dass die Datenrate während der Datenübertragung (*Data Phase*) flexibel ist. Das bedeutet, dass die Übertragungszeit pro Bit in diesem Bereich variabel verkürzt werden kann, um höhere Geschwindigkeiten zu erzielen.

### 2.3.3 LIN

Als kostengünstigere Alternative zu CAN-Netzwerken wurde gegen Ende der 1990er Jahre der **Local Interconnect Network (LIN)**-Bus entwickelt. LIN wurde unter ISO 17987-1 bzw. **Society of Automotive Engineers (SAE) J2602** spezifiziert.

Der LIN-Bus ist Bereichen, in denen keine hohen Bandbreiten, geringe Latenzzeiten oder aufwändige Konfigurationsmöglichkeiten benötigt werden angesiedelt. Ein Anwendungsbereich ist beispielsweise die Übertragung von Daten des Schließsystems, der elektronischen Fensterheber und der Sitzelektronik. Die maximale Bitrate liegt bei 20Kbit/s. Das Ziel

der LIN-Einführung bestand darin, dass die Implementierung eines LIN-Knotens kosten-effizienter durchzuführen ist als jene eines CAN-Knotens.

LIN wird oft als Sub-Bus von CAN verwendet, um einen gewissen Grad an Kompatibilität und Vernetzung zu gewährleisten. Dabei existiert ein entsprechendes Gateway zur Verbindung von CAN und LIN.

Folgendes Schaubild veranschaulicht den Aufbau (angelehnt an [Wer14] S. 81):

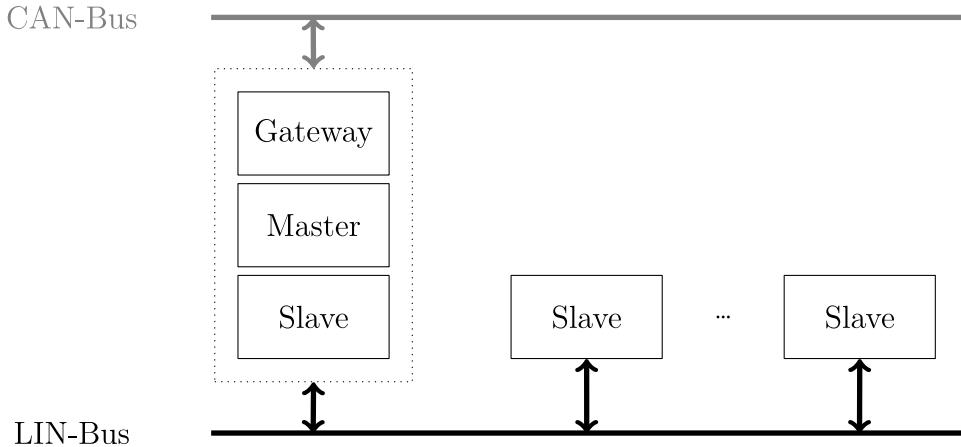


Abbildung 14: Fahrzeugprotokolle: LIN: Busaufbau

Der Master-Knoten ist für den Kommunikationsablauf, die Konfiguration und das Zeitmanagement des Netzwerks verantwortlich. Slaves antworten auf Anfragen des Masters, welche als Broadcast versendet werden. Slaves entscheiden somit selbst, wie für Busnetzwerken typisch, ob auf eine Anfrage reagiert werden soll. Hierbei findet die Kommunikation auf einer einzelnen Datenleitung statt, welche auf K-Line (ISO 9141) basiert [lis].

Der LIN-Bus verwendet wie der CAN-Bus eine inhaltsbezogene Adressierung (Abbildung angelehnt an [Wer14] S. 82):

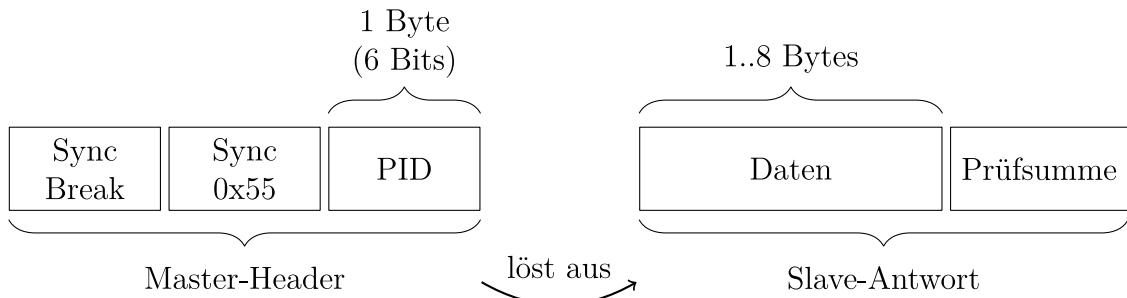


Abbildung 15: Fahrzeugprotokolle: LIN: Paketformat

Die Bitsequenz Sync Break besteht aus  $\geq 13$  Low-Bits (0), auf welche ein High-Bit (1) folgt. Dies markiert den Anfang einer Nachricht. Das anschließend übertragene Synchronisations-Byte hat den fixen Wert 0x55. Binär kodiert ergibt sich hieraus eine alternierende Folge von 0- und 1-Bits. Durch das **Parameter Identifier (PID)**-Byte wird

die Bedeutung der Nachricht festgelegt. Die PIDs von 0x00 bis 0x3B sind für beliebige Bedeutungen reserviert und werden in einer entsprechenden Konfigurationsdatei festgelegt. Die zwei übrigen Bits innerhalb des PID-Bytes stellen Paritätsbits dar.

Durch die Versendung eines Master-Header-Pakets wird die Antwort eines angesprochenen LIN-Knotens an den Master-Knoten ausgelöst.

Die Diagnose von LIN-Slaves kann mittels ISO-TP und UDS erfolgen [lem]. Dafür verwendete Paketformate werden von den verwendeten Protokollen definiert. Der Master empfängt im Rahmen einer Diagnose entsprechende Diagnose-Frames, eventuell aus anderen Busnetzwerken, und leitet dazu passende Anfragen an zu diagnostizierende Slaves weiter. Dabei sendet der Master einen Diagnose-Request mit der PID 0x3C und erhält eine Diagnose-Response, in welcher die PID 0x3D enthalten ist.

## 2.4 KWP2000

Ein weit verbreitetes Protokoll auf Applikationsebene stellt KWP2000 dar. In einem vorherigen Abschnitt (2.3.1) wurde bereits das Zusammenspiel dieses Protokolls mit dem K-Line-Bus beschrieben. In diesem Abschnitt folgen die Aspekte bezüglich der Diagnosefunktion und generelle Informationen zu Diensten, welche durch KWP2000 auf Applikationsebene angeboten werden.

Über die Nutzdaten von KWP2000-Paketen (siehe Abbildung 5) können Nachrichten auf der Anwendungsschicht übertragen werden. In der Praxis wird oft das ISO-TP-Transportprotokoll verwendet, um KWP2000-Pakete zu versenden. Dies hat den Vorteil, dass die Paketgröße nicht auf 255 Bytes beschränkt wird.

Folgendes Schema verdeutlicht den Aufbau der in den Nutzdaten eingebetteten Botschaften:

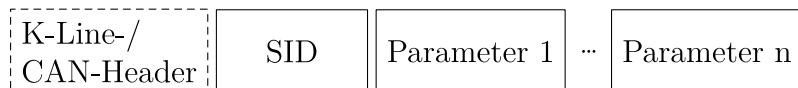


Abbildung 16: Fahrzeugprotokolle: KWP2000: Paketformat auf Anwendungsschicht

Der erste Teil des Pakets besteht aus den Kopfdaten des darunterliegenden Busprotokolls. Hierfür kommen K-Line und CAN infrage. Das SID-Byte wählt den zu kontaktierenden Dienst aus. Hierfür gibt es sowohl standardisierte Tabellen für gesetzlich vorgeschriebene Services als auch nichtöffentliche SIDs der Fahrzeughersteller für proprietäre Dienste.

Ein Überblick über standardisierte PID-Bereiche kann anhand folgender Tabelle [Wer14] verschafft werden:

Tabelle 4: Fahrzeugprotokolle: KWP2000: Service-IDs

Servicetyp	Request	Positive Reponse	Negative Response	Standard
OBD-Dienste	0x00 - 0x0F	0x40 - 0x4F	0x7F00 - 0x7F0F	SAE J1979 ISO 15031-5
Allg. Dienste für K-Line und CAN	0x10 - 0x3E	0x50 - 0x7E	0x7F10 - 0x7F3E	ISO 14230-3
Escape-Codes	0x80	0xC0	0x7F80	ISO 14230-3
K-Line-Dienste	0x81 - 0x83	0xC1 - 0xC3	0x7F81 - 0x7F82	ISO 14230-2
CAN-Dienste	0x84 - 0x85	0xC4 - 0xC5	0x7F84 - 0x7F85	ISO 15765-3
Herstellerspez. Dienste	0xA0 - 0xBE	0xE0 - 0xFE	0x7FA0 - 0x7FBE	Fahrzeug-/ Systemhersteller

Alle nicht aufgeführten Werte gelten als reserviert.

KWP2000 sieht verschiedene Kategorien von Diagnosediensten vor. Für eine Sicherheitsanalyse sind vor allem folgende Dienstkategorien interessant:

- Upload/Download: Beliebiger Lese- und Schreibzugriff auf Steuergerätespeicher. Dies erlaubt unter anderem das Aktualisieren von Steuergeräten mittels modifizierter Software
- Remote Activation of Routines: Beliebige auf Steuergeräten gespeicherte Routinen können ausgeführt werden
- Data Transmission: Lese- und Schreibzugriff auf Datensätze von Steuergeräten
- Input/Output Control: Ein- und Ausgänge der Steuergräte ansteuern

Diese Dienste sind jedoch nicht für jedes angeschlossene Diagnosegerät verfügbar. Fahrzeugherrsteller können mittels KWP2000 Sicherheitsmechanismen nutzen, um den Zugriff auf sensible Dienste zu beschränken. Hierzu sind mehrere verschiedene Diagnosemodi spezifiziert, welche unterschiedliche Arten von Zugriffen erlauben. Standardmäßig befinden sich alle Steuergeräte im Grundzustand (Default Diagnostic Session). Hiermit können nur wenige und für die Analyse eher unwichtige Dienste angesprochen werden. Folgende standardisierte Modi sind verfügbar:

Tabelle 5: Fahrzeugprotokolle: KWP2000: Session-Modi

Request	Typ	Anmerkung
0x81	Default Diagnostic Session	Standardzustand
0x85	Programming Session	Programm- und Datenspeicher lesen bzw. schreiben
0x86	Development Session	Entwicklermodus
0x87	Adjustment Session	Anpassung der Geräte- Parameter
0x89 - 0xFE	Hersteller spezifische Sessions	

Um höhere Privilegien zu erhalten, kann eine Rechteausweitung mittels der Start-Diagnostic-Session-Nachricht (0x10) gestartet werden. Die genauen Zugriffsbedingungen sind von den Automobilbauern festzulegen und sind nicht öffentlich einsehbar. Generell muss für eine Rechteausweitung ein Schlüsselaustausch zwischen Diagnosetestgerät und Steuergerät erfolgen. Hierfür ist der Nachrichtentyp Security Access (0x27) verfügbar, welcher zwei Parameter verwendet. Der erste Parameter, der Request-Seed, wird von einem Steuergerät an das Testgerät gesendet. Aus diesem Seed muss der Empfänger den dazugehörigen Schlüssel berechnen und zurück an das Steuergerät senden. Bei Erfolg findet eine Freischaltung für den gewünschten Diagnosemodus statt. Weitere Parameter und Abwandlungen sind bei diesem Prozess jedoch möglich und werden durch die Fahrzeughersteller möglicherweise abweichend implementiert. Beispielsweise sind Seed- und Schlüsselerzeugung *nicht* spezifiziert und werden somit als proprietäre Algorithmen implementiert. Denkbar ist auch die Verwendung verschiedener Schlüsselmechanismen für Hersteller und Vertragswerkstätten. Ein weiterer Faktor, welcher für die Verfügbarkeit von einigen Diensten entscheidend sein kann ist zum Beispiel der Status der Zündung.

Bei erfolgter Freischaltung muss in einem Intervall von fünf Sekunden eine Tester-Present-Nachricht mit der SID 0x3E versendet werden. Ansonsten erlischt aus Sicherheitsgründen die Freigabe.

Zur Informationsgewinnung über das verbundene Steuergerät kann das SID-Byte 0x1A verwendet werden. Diese SID beschreibt *Read ECU Identification*-Pakete, mittels welchen identifizierende Informationen von einem Steuergerät angefordert werden können. Diese Informationen sind vor allem bei der Analyse auf Hardware-Ebene hilfreich.

Für physikalische Adressen gibt es Vorschläge aus der ISO 14230-2-Norm. Diese besagen, dass Diagnosetester den Adressbereich 0xF0 - 0xFD, Motorsteuergeräte 0x10 - 0x17, Getriebesteuergeräte 0x18 - 0x1F und ABS bzw. ESP den Bereich 0x28 - 0x2F verwenden *sollten*. Ein Zwang dies aus der Seite der Hersteller zu befolgen existiert jedoch nicht. Die funktionale Adresse 0x33 wird implementiert, um den gesetzlichen Anforderungen zu entsprechen: Alle Steuergeräte können über diese Adresse angesprochen werden [emoa].

Eine Liste der standardisierten Service-IDs für diverse Dienste kann [Wer14] entnommen werden. Beispiele hierzu sind der direkte Speicherzugriff, beliebiges Ausführen von in Steuergeräten implementierten Routinen und das Auslesen bzw. Programmieren der Firmware:

Service	PID	Dienstbezeichnung
Read Data By Local Identifier	0x21	Auslesen von Steuergerätedaten via ID oder Speicheradresse + Länge
Read Data By Common Identifier	0x22	
Read Memory By Address	0x23	
Write Data By Common Identifier	0x2E	Schreiben in Speicher von Steuergeräten via ID oder Speicheradr.
Write Data By Local Identifier	0x3B	
Write Memory By Common Identifier	0x3D	
Input Output Control By Common Identifier	0x2F	Feste Werte von Ein- und Ausgängen setzen
Input Output Control By Local Identifier	0x30	
Request Download	0x34	Download der Firmware
Request Upload	0x35	Upload von Firmware

Weitergehend hierzu können proprietäre PIDs mittels Reverse-Engineering und Fuzzing ermittelt werden, sofern der Mechanismus zum Erlangen von zusätzlichen Privilegien kein Hindernis darstellt. Eine vom Standard definierte Möglichkeit der Interaktion mit herstellerspezifischen Diensten ist der Escape Code Request. Die Bedeutung aller darin enthaltenen Nutzdaten ist herstellerabhängig. Der Funktionsumfang undokumentierter und proprietärer Services kann vielfältig sein – von der Diagnose bis hin zu sicherheitskritischen Einstellungen ist jegliches Ausmaß möglich.

#### 2.4.1 UDS

Um eine herstellerunabhängige Art und Weise der Diagnose zu gewährleisten [Smi16], wurde UDS (Unified Diagnostic Services) entwickelt und unter ISO 14229 spezifiziert. Dabei baut UDS auf KWP2000 auf und verändert bzw. erweitert Aspekte des Basisprotokolls. UDS verwendet meist ISO-TP als Transportschicht [Smi16].

Eine Erweiterung auf Paketbasis stellt die Einführung des LEV-Byte (Subfunction Level) dar:



Abbildung 17: Fahrzeugprotokolle: UDS: Paketformat

Bei der Spezifikation von UDS wurden diverse SIDs von KWP2000 zu einer UDS-SID zusammengefasst. Die gewünschte Unterfunktion einer zusammenfassenden UDS-SID kann mittels LEV-Byte gewählt werden.

Während bei KWP2000 Dienste und Daten via ID oder Speicheradresse angesprochen werden können, ist bei UDS nur eine Adressierung mittels 16-Bit-IDs vorgesehen. **Es ist daher bei Analysen zu Prüfen, ob KWP2000- oder UDS-SIDs verwendet werden müssen.**

Ein weiterer Unterschied liegt in der Nummerierung der verschiedenen Session-Modi. Die Nummerierung startet bei 0x01 für die Default Session und wird für jeden Eintrag um den Wert 0x01 inkrementiert. Neu ist die explizite Festlegung, welche Dienste bei einer Standard-Diagnosesitzung angeboten werden müssen. Hierzu zählen Dienste wie das Auslesen des Fehlerspeichers und das Zurücksetzen von Steuergeräten. Interessant ist zudem, dass das Lesen und Schreiben von Steuergerätedaten und das Ausführen von Routinen auf dem Steuergerät über eine Standard-Diagnosesitzung erlaubt wird. Der Hersteller kann jedoch explizit festlegen auf welche Bereiche und Daten ohne höhere Rechte zugegriffen werden darf.

Durch ISO 15764 wird eine mögliche Transportverschlüsselung für Busdaten spezifiziert. UDS erlaubt den Einsatz einer derartigen Verschlüsselung als zusätzliche Schicht zwischen den Anwendungsschichten von Diagnosetestgerät und UDS-Anwendungsschicht im Fahrzeug. Verschlüsselte Pakete (Secured Data Transmission) werden mit der SID 0x84 gekennzeichnet.

## 2.5 Schnittstellen von Automobilnetzwerken

Um Daten aus Fahrzeugnetzwerken aufzuzeichnen, zu manipulieren oder um Daten in ebensolche einzuschleusen gibt es prinzipiell mehrere Möglichkeiten. Vor- und Nachteile und die Eignung für bestimmte Zwecke bilden das Thema dieses Abschnitts.

### 2.5.1 OBD-II

Laut einer EU-Richtlinie des europäischen Parlaments und des europäischen Rates ist eine Schnittstelle zur On-Board-Diagnose für alle Fahrzeuge verpflichtend [euro] (Erwägungsgrund 14). Diese europäische OBD-Norm (EOBD) schreibt somit vor, dass alle PKWs mit Ottomotor ab Modelljahr 2001 und alle Fahrzeuge mit Dieselmotor ab Modelljahr 2004 in der EU zwingend über eine solche Diagnoseschnittstelle verfügen müssen. Diese Maßnahme wird ergriffen, um auf eine standardisierte Art und Weise Emissionswerte von Fahrzeugen zu ermitteln und eventuellen Anomalien entgegenzuwirken. Entsprechende Vorschriften gelten auch auf dem US-Markt (US-OBD).

Folgende Abbildungen [obda] [obdb] zeigen schematisch die weiblichen Steckverbinder vom Typ A und B des OBD-II-Standards. Der Steckertyp A wird bei PKWs mit 12V Betriebsspannung eingesetzt, wobei Typ B bei Nutzfahrzeugen mit 24V Spannung den Einsatz findet [obdt]. Die Spezifikation des Steckerdesigns erfolgt nach der Norm SAE J1962.

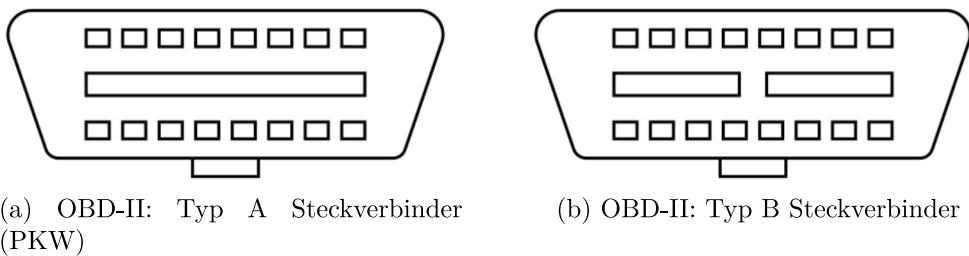


Abbildung 18: OBD-II: Typen von Steckverbindern

Diese Unterscheidung verhindert die Beschädigung von Diagnosehardware, wenn versehentlich der Versuch unternommen wird ein für 12V ausgelegtes Testgerät mit einem 24V-OBD-Port zu verbinden.

Die Position der Steckerbuchse ist fahrzeugabhängig – meist befindet sie sich in der Nähe des Fahrersitzes oder unter einer Verkleidung im Bereich des Fußraumes. Entsprechende Hardware, welche zur Interaktion mit dieser Fahrzeug-Schnittstelle verwendet werden kann, wird im Abschnitt [3.1.1](#) vorgestellt.

Folgende Abbildungen zeigen die Pinbelegungen [\[Smi16\]](#) von gängigen OBD-II-kompatiblen Protokollen:

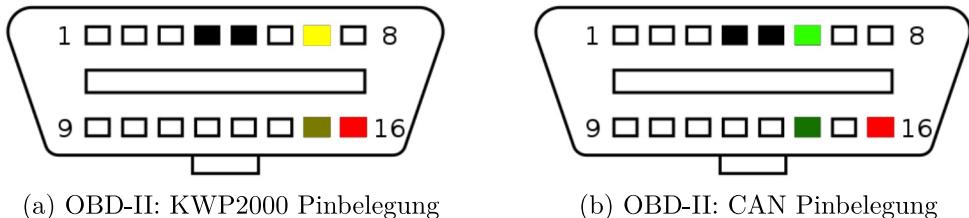


Abbildung 19: OBD-II: Pinbelegungen

Tabelle 6: OBD-II: Pinbelegungen: Legende

Farbe	Bedeutung
■	Fahrzeugmasse (4), Signalmasse (5)
■	+12V/4A / +24V/2A
■	K-Line
■	L-Line (optional)
■	CAN-High
■	CAN-Low

Darüber hinaus existieren Pins, welche Hersteller beliebig belegen und verwenden können [\[Smi16\]](#):

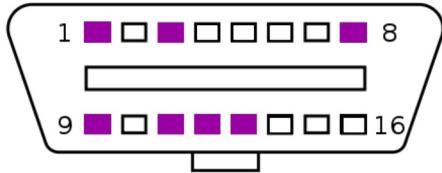


Abbildung 20: OBD-II: Herstellerspezifische Pinbelegungen

Diese Pins können beispielsweise zur Interaktion mittels proprietärer Protokolle oder zur Übertragung einzelner Datenwerte, wie Motordrehzahl oder Zündungsstatus verwendet werden.

Nicht jedes im vorhergehenden Abschnitt angesprochene Protokoll ist, selbst wenn es intern im Fahrzeug verwendet wird, über den OBD-II-Anschluss verfügbar. Dies liegt darin begründet, dass über die Diagnoseschnittstelle OBD-II lediglich eine Interaktion mit Diagnoseprotokollen vorgesehen ist. Oft sind somit nur die Leitungen für CAN und K-Line über den OBD-II-Port exponiert. Der Zugriff wird zusätzlich über ein Gateway gefiltert und verwaltet (siehe 2.2). Das bedeutet, dass im Regelfall durch simples Aufzeichnen von Daten über den OBD-Port keine oder nur weniger für die Analyse relevante Daten gesammelt werden können. Ohne vorherige Anfrage von Steuergeräten werden meist keine oder nur wenige Daten an die OBD-Schnittstelle versandt. *Prinzipiell* ist zudem das Filtern von beliebig eingeschleusten Datenpaketen durch das Gateway möglich.

Das erste Byte des Nutzdatenbereichs stellt bei der Verwendung von OBD-II das Mode-Byte dar. Mit diesem wird die Art der Anfrage angegeben. Es sind unter anderem Modi zur Abfrage von Daten und für das Management des Fehlerspeichers vorgesehen, welche aus [Smi16] (S. 58) oder aus öffentlich verfügbaren Listen entnommen werden können. Beispielsweise kann mittels Modus 0x09 ermittelt werden, welche Diagnose-PIDs unterstützt werden. Ein interessanter Modus, 0x08, stellt mithilfe von proprietären PIDs Funktionen zur manuellen Kontrolle von internen Systemen für Techniker bereit [Smi16]. Die Durchführung von Manipulationen mit diesem Modus kann jedoch nicht gänzlich ausgeschlossen werden, weshalb dieser Modus oft ein Ziel von Sicherheitsanalysen darstellt.

Unter [Wer14] (S. 216ff) befindet sich eine aufgearbeitete Liste mit nützlichen Standard-PIDs. Informationen zur Konvertierung von Antwortpaketen in konkrete Nutzdaten können dort ebenfalls entnommen werden. Neben den öffentlich verfügbaren PIDs existieren oft proprietäre PIDs als Teil der herstellerspezifischen K-Matrix.

### 2.5.2 Abzweigung von Busleitungen

Über den OBD-II-Anschluss ist, wie bereits beschrieben, meist nur der Diagnosebus erreichbar. Dieser ist meist über ein Gateway von den sonstigen Busnetzen abgeschirmt. Um dennoch für eine Sicherheitsanalyse relevante Daten abzugreifen oder in den Bus einzuschleusen, müssen Busleitungen oft direkt als Schnittstelle dienen. Hierzu werden Abzweigungen der Datenleitungen geschaffen, um mittels geeigneter Hardware ungefiltert Daten aus dem Datenbus zu lesen oder um beliebige Pakete einzuschleusen. Zunächst müssen jedoch die dafür benötigten Leitungen im Fahrzeug ausfindig gemacht werden. Je nach Protokoll und verwendeter Technik müssen verschiedene Arten von Leitungen lokalisiert

werden. Die Charakteristika des Zielbusses können bei der Lokalisierung von Busleitungen helfen. Folgendes Beispiel gilt für den oft analysierten CAN-Bus:

- Der CAN-Bus besteht aus einem verdrillten Kabelpaar
- Das Bussystem verfügt über Abschlusswiderstände. Das bedeutet, dass je nach Verkabelung auf potentiellen Leitungen ein Widerstand von entweder  $120\Omega$  oder  $60\Omega$  gemessen werden kann [asi].
- Die Datenübertragung erfolgt über differentielle Signalübertragung mit einer Spannung zwischen 2,5V und 3,5V [Smi16].

Auf folgendem Bild wurde der CAN-Bus anhand des gemessenen Widerstands lokalisiert [esd]:



Abbildung 21: Netzwerkschnittstellen: Lokalisierung von CAN-Leitungen

Oft können auch Einbauanleitungen für Alarmanlagen von Drittanbietern bei der Lokalisierung von CAN-Leitungen helfen [esd], da die angebotenen Sicherheitslösungen den

CAN-Bus verwenden. In den Anleitungen werden demnach bevorzugte Einbauorte, Busverkabelungen und die Farben der gesuchten Leitungen genannt. Eventuell kann eine CAN-Verbindung hinter dem Radio vorgefunden werden, da moderne Radiosysteme zur Bereitstellung von Funktionen CAN-Daten benötigen.

Um Daten direkt aus dem Bussystem abzugreifen gibt es diverse Möglichkeiten. Die naheliegendste, aber auch risikoreichste Methode ist das Durchtrennen der Busleitungen, um zusätzliche Hardware in den Bus einzubringen. Es besteht jedoch das Risiko, dass der Bus somit unbrauchbar wird. Aus diesem Grund bietet sich die Verwendung spezieller Abzweigverbinder an. Abgezweigte Leitungen werden bei der Verwendung dieser Methode nicht komplett durchtrennt, sodass ein normaler Betrieb des Fahrzeugbusses nicht gefährdet wird.

## 3 Analyse von Fahrzeugnetzwerken

In diesem Kapitel werden die nötigen Voraussetzungen und Methoden erläutert, um Fahrzeugnetzwerke zu analysieren. Zunächst werden gängige Hard- und Softwarewerkzeuge vorgestellt. Anschließend werden physische und virtuelle Testumgebungen vorgestellt, mittels welchen die Analyse von Fahrzeugen näher gebracht wird. Gängige Techniken und Vorgehensweisen des Reverse-Engineerings schließen dieses Kapitel ab.

### 3.1 Vorbereitung

Je nach Anwendungsgebiet und gewählten Analysemethoden müssen verschiedene vorbereitende Maßnahmen getroffen werden, bevor eine Untersuchung durchgeführt werden kann. Im Rahmen dieses Abschnitts werden hierzu hilfreiche Aspekte aus den Bereichen der Hard- und Software vermittelt.

#### 3.1.1 Hardware

In diesem Abschnitt werden ausgewählte Hardware-Lösungen zur Analyse von Fahrzeugnetzwerken vorgestellt. Der Fokus liegt hierbei auf kostengünstigen Geräten, welche zum Teil als Open-Source-Projekte entwickelt werden.

Grundsätzlich kann zwischen zwei Adaptertypen unterschieden werden:

1. Adapter zur Kommunikation mit OBD-Schnittstellen
2. Adapter zur Verwendung beliebiger Busleitungen

Der erste Adaptertyp wird meist in Werkstätten zur Durchführung von Diagnose- und Aktualisierungsvorgängen verwendet. Optional ist die Verwendung eines Bluetooth-Dongles mit identischem Funktionsumfang möglich. Derartige Geräte sind bereits für unter 20€ verfügbar. Alle Angriffe, welche über die OBD-II-Schnittstelle durchgeführt werden können, werden mit diesen Adapters durchgeführt.

Einen entsprechenden Adapter stellt *USB2CAN* dar, welcher von 8devices entwickelt wurde [dev]:



(a) Konverter mit USB-Kabel



(b) D-Sub DE9-Verbindung

Abbildung 22: Hardware: USB2CAN

Die Schemata, Diagramme und Linux-Treiber hierzu sind Open Source. Hinzu kommt eine Programmierschnittstelle zur Verwendung der Hardware in eigenen Softwareprojekten.

Über ein USB-Kabel erfolgt die Verbindung mit einem Rechner. Via D-Sub DE9 wird über ein OBD-Kabel die Kommunikation mit dem Fahrzeug hergestellt. *USB2CAN* beherrscht das vollständige CAN-Protokoll (11-Bit- und 29-Bit-IDs) mit vielen verschiedenen Busgeschwindigkeiten und kann über Firmware-Updates aktualisiert werden.

*Hinweis:* Vor der Benutzung des CAN-Adapters muss das dazugehörige Kernel-Modul [[dev](#)] kompiliert und geladen werden. Dieser Vorgang muss unter Umständen erneut durchlaufen werden, wenn ein Kernel-Update vorgenommen wurde.

*USB2CAN* kann, wie auf Abbildung 29 zu sehen, über spezielle Adapter auf einzelne Busleitungen zugreifen. Dies ist nicht bei allen Adapters desselben Typs aufgrund von fest montierten Kabeln möglich und ist im Vorfeld zu prüfen.

Neben *USB2CAN* gibt es diverse weitere Adapter [[jtc](#)], welche verschiedene weitere Protokolle unterstützen.

Als Alternative zu den vorgestellten Adapters dienen Geräte, welche mit beliebigen Busleitungen verbunden werden können. Ein Anwendungsszenario umfasst somit das Abzweigen und die Analyse von CAN-Daten über einen derartigen Adapter. Auf gleiche Art und Weise können CAN-Pakete in den Bus eingefügt werden, ohne dass ein filterndes Gateway die Interaktion verhindert.

Zwei Open Source Entwickler-Boards, welche diesen Zweck erfüllen, sind *CANtact* [[eve](#)] und *USBtin* [[fisb](#)] [[hwh](#)] [[fisa](#)]:

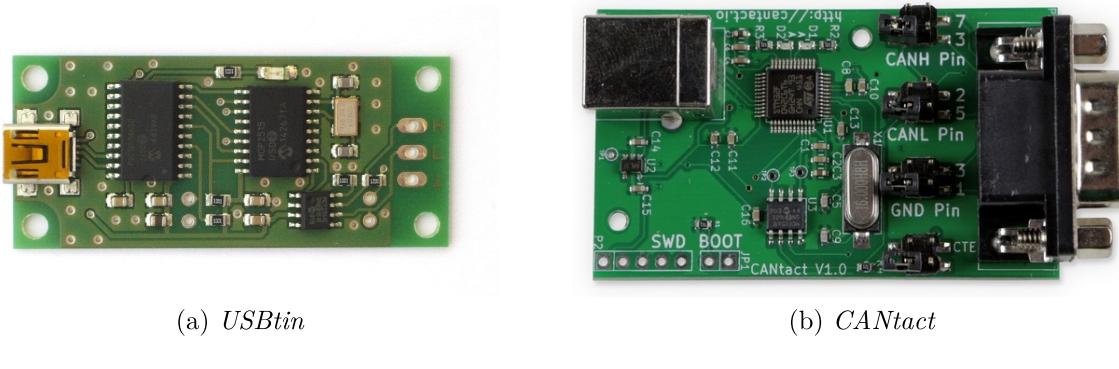


Abbildung 23: Hardware: *USBtin* und *CANtact*

Beide Platinen erlauben das direkte Anschließen von CAN-Leitungen, um ungefiltert Pakete zu lesen und zu schreiben. Vor der Verwendung ist, je nach erworbener Ausführung, das Verlöten von Mikrocontrollern oder Kabelverbünden vonnöten.

Beim Kauf von Analysehardware ist stets auf die Liste der unterstützten Protokolle zu achten. Oftmals wird das sehr weit verbreitete CAN-Protokoll unterstützt. Für die Unterstützung von beispielsweise K-Line muss auf spezielle Hardware geachtet werden. Bluetooth-Adapter unterscheiden sich aufgrund der abweichenden Anbindung von USB-Adapters. Während Bluetooth-Adapter meist für die Nutzung mit spezieller Diagnose-Software oder

Android-Geräten ausgelegt sind, werden die USB-Varianten meist im Zuge von Busanalysen verwendet. Die Nutzung von Bluetooth-Dongles als virtuelles serielles Interface ist möglich, die Softwareunterstützung kann jedoch nicht garantiert werden.

Neben eigenständigen Dongles und Adapters werden darüber hinaus OBD- und Busmodule für *Raspberry Pi*- und Arduino-Geräte angeboten. Dies erlaubt eine flexible Ansteuerung, zum Beispiel über **General Purpose Input/Output (GPIO)**, und den platzsparenden Betrieb innerhalb von Fahrzeugen. Eine Einsatzmöglichkeit besteht im Aufbau eines Geräts zur kontinuierlichen Datenaufzeichnung, welches Daten während einer Fahrt oder Aktion aufzeichnet. Diese Daten können im Anschluss analysiert werden.

Neben der Verwendung von Standard-OBD-Kabeln ist die Nutzung von weiteren Kabetypen [oca] [ocy] möglich:

verwendet. Über die freigelegten Leitungsstücke ist das Einbringen von Analysehardware möglich, welche über keinen festen OBD-Verbinder verfügt.

Mit einem weiteren Open-Source-Projekt namens *CANBadger* [jhca] können vielfältige und neuartige Attacken durchgeführt werden. Alle Spezifikationen und Softwareteile des Boards sind öffentlich verfügbar, was einen Nachbau und die Anpassung des Projekts ermöglicht.

Mittels zweier CAN-Schnittstellen sind *Man In The Middle (MITM)*-Szenarien in Echtzeit durchführbar. Daten können so ohne nennenswerte Verzögerungen auf eine SD-Karte aufgezeichnet und beliebig manipuliert werden.

Darüber hinaus kann eine neuartige Attacke auf UDS-Sitzungen durchgeführt werden: Durch einen Man-in-the-Middle-Angriff zwischen einem Fahrzeug und einem Diagnose-Testgerät kann eine durch den Diagnosevorgang ausgehandelte privilegierte Diagnose-Sitzung übernommen werden. Hierzu muss *keinerlei* Wissen über den Authentifizierungsmechanismus vorhanden sein. Während der Authentifizierung überwacht der *CANBadger* den Verbindungsauftbau und trennt bei Erfolg das offizielle Testgerät des Herstellers vom Fahrzeugbus. Die verbleibende Diagnosesitzung kann mit den gewünschten Rechten weiterhin betrieben werden, um beispielsweise den Speicher von Steuergeräten auszulesen.

Die Busgeschwindigkeit wird durch den *CANBadger* automatisch erkannt und das Gerät ist über eine Server-Applikation via Ethernet aus der Ferne steuerbar. In Kombination mit einer optionalen Batterie kann der *CANBadger* hierdurch fest in Fahrzeugen verbaut und aus der Ferne für Angriffe genutzt werden. Unterstützt wird das CAN-Protokoll mit UDS und TP 2.0. Das gesamte Board kann *theoretisch* für unter 25\$ konstruiert werden [jhcb]. Je nach Komponentenverfügbarkeit und erworbener Stückzahl kann dieser Preis jedoch sehr schwanken.

Durch die Verwendung des vorgestellten Werkzeugs können zusammenfassend folgende Analysemethoden realisiert werden:

- Eingriffe in die Kommunikation zwischen Bussystemen und Steuergeräten
- Aufzeichnung des Netzwerkverkehrs
- Manipulation von Fahrzeugen aus der Ferne mittels der Server-Applikation
- Angriffe auf UDS

Durch die Flexibilität des Werkzeugs ist zudem die Anpassung auf spezielle Szenarien möglich.

### 3.1.2 Software

Im folgenden Abschnitt werden alle benötigten Pakete und Befehle vorgestellt, um mit einem linuxbasierten Betriebssystem eine Verbindung zu einem Automobilnetzwerk herzustellen. Der Fokus liegt auf dem CAN-Netzwerk. Anschließend erfolgt eine Übersicht über diverse nützliche Analysewerkzeuge.

In den Standard-Paketquellen existiert das `can-utils`-Paket. Dies ist ein Open-Source-Projekt [cut], welches die Kommunikation mit Fahrzeugnetzwerken über Sockets zulässt. Eigens entwickelte Applikationen können dadurch über einen Socket des Typs `PF_CAN` generisch und unabhängig von der eingesetzten Hardware mit Fahrzeugen interagieren [Smi16]. Diese Socket-Verbindungen werden als *SocketCAN* bezeichnet.

Sobald `can-utils` installiert und ein unterstütztes USB-Gerät mit dem Rechner verbunden wurde, können folgende Ausgaben im Systemlog nachvollzogen werden:

```
1 usb 2-1.2: new full-speed USB device number 4 using ehci-pci
2 usb 2-1.2: New USB device found, idVendor=0483, idProduct=1234
3 usb 2-1.2: New USB device strings: Mfr=1, Product=2, SerialNumber=3
4 usb 2-1.2: Product: USB2CAN converter
5 usb 2-1.2: Manufacturer: edevices
6 usb 2-1.2: SerialNumber: ED000200
7 usb_8dev 2-1.2:1.0 can0: firmware: 1.5, hardware: 1.0
8 manager: (can0): new Generic device [...]
```

Listing 1: Software: Systemlog-Ausgabe bei der Verwendung von USB2CAN

Zusätzlich kann der CAN-Adapter bei Erfolg über `ifconfig -a` als Netzwerk-Interface aufgelistet werden.

Bevor das Interface verwendet werden kann, muss zunächst die Bitrate des Interfaces derer des angeschlossenen Bussystems gleichen. Folgender Befehl setzt die Bitrate und aktiviert das Interface:

```
1 sudo ip link set <CAN-Iface> up type can bitrate <Bitrate in Bit/s>
```

Listing 2: Software: Einstellung der Bitrate

Im Falle von High-Speed-CAN beträgt die Bitrate meist 500 Kb/s. Low-Speed-CAN verwendet dagegen meist 250 Kbit/s oder 125 Kbit/s [Smi16]. Sollte die Bitrate nicht korrekt sein, werden keine intakten Pakete empfangen.

*Hinweis: Im Falle von seriellen CAN-Adaptoren muss das Programm slcand zur Adapter-Konfiguration verwendet werden.*

Ein erster Test auf empfangene Pakete kann anschließend wie folgt durchgeführt werden:

```
1 candump <CAN-Interface>
```

Listing 3: Software: Starten von candump

Sofern alle Einstellungen und die Verkabelung korrekt sind, können CAN-Pakete empfangen und gespeichert werden:

```
1 can0 136 [8] 00 02 00 00 00 00 00 00 1B
2 can0 13A [8] 00 00 00 00 00 00 00 00 19
3 can0 13F [8] 00 00 00 05 00 00 00 00 1F
4 can0 164 [8] 00 00 C0 1A A8 00 00 00 31
5 can0 17C [8] 00 00 00 00 10 00 00 00 12
6 can0 18E [3] 00 00 5C
```

```

7 | can0 039 [2] 00 0C
8 | can0 183 [8] 00 00 00 0A 00 00 10 10
9 | can0 143 [4] 6B 6B 00 D1
10 | can0 095 [8] 80 00 07 F4 00 00 00 08

```

Listing 4: Software: Ausgabe von candump

In der zweiten Spalte werden die IDs der CAN-Pakete aufgelistet. Es folgt eine Angabe über die Anzahl der enthaltenen Nutzdaten-Bytes. Im Anschluss werden die enthaltenen Nutzdaten ausgegeben.

Sollten keine Pakete empfangen werden, können folgende Maßnahmen zur Diagnostizierung des Problems durchgeführt werden:

- Überprüfung der Verkabelung
- Auslösen von Aktionen, wie z.B.: Öffnen/Schließen von Türen, Betätigen des Fernlichtes, Wechsel des Radiosenders
- Überprüfung des Systemlogs

Beim Empfang sehr vieler Pakete, was in der Praxis auf einem Fahrzeugbus nicht unüblich ist, kann bei der Verwendung von `candump` keine Übersicht gewährleistet werden. Aus diesem Grund wurde `cansniffer` entwickelt. Das Ziel dieser Software ist die Gruppierung der erhaltenen Pakete nach CAN-ID. Ändernde Bytes im Nutzdatenbereich können bei Bedarf farblich hervorgehoben werden. Es ist zudem möglich, dass Pakete mit identischer CAN-ID, welche keine ändernden Daten enthalten, automatisch gefiltert werden. Über interaktive Funktionen können dynamisch Optionen angepasst und grundlegende Filterfunktionen verwendet werden.

Neben `candump` und `cansniffer` sind im Paket `can-utils` weitere Anwendungen enthalten. Die zentralen Komponenten aus diesem Paket umfassen:

Tabelle 7: Software: Wichtige Komponenten aus can-utils

Name	Funktion
candump	Auflistung / Speicherung von empfangenen CAN-Paketen mit optionalen Filtern
cansniffer	Gruppiert empfangene Pakete nach ID Erkennt sich ändernde Bytes in Paketen mit gleichen IDs Filtert Pakete, welche über längere Zeit nicht mehr erhalten wurden
cansend	Versendet das angegebene CAN-Paket
canplayer	Versendet die in einer Datei angegebenen Pakete
cangen	Generierung von (zufälligen) CAN-Paketen Versendung der Pakete in Intervallen
isotop*	Viele für CAN verfügbare Anwendungen sind in can-utils als ISO-TP-Variante enthalten

### **Wireshark:**

Die Netzwerk-Analysesoftware *Wireshark* kann mithilfe von *SocketCAN* zum analysieren und speichern von CAN-Daten benutzt werden:

```
▶ Frame 10298: 16 bytes on wire (128 bits), 16 bytes captured (128 bits) on interface 0
└─ Controller Area Network
    . . . 0 0000 0000 0000 0001 0110 0001 = Identifier: 0x00000161
    0... .... .... .... .... .... .... = Extended Flag: False
    .0.. .... .... .... .... .... .... = Remote Transmission Request Flag: False
    ..0. .... .... .... .... .... .... = Error Flag: False
    Frame-Length: 8
└─ Data (8 bytes)
    Data: 000005500108002b
    [Length: 8]
```

Abbildung 25: Software: CAN-Sniffing mit *Wireshark*

In eigenen Tests konnte gezeigt werden, dass falls die Programme aus dem `can-utils`-Paket Anomalien vorweisen, Wireshark als hilfreiche Alternative dienen kann.

### **Metasploit:**

Das Framework für Penetration-Tests namens *Metasploit* wurde Anfang 2017 um Funktionen erweitert, mit denen eine Analyse von Fahrzeugnetzwerken möglich ist [urh]. Eine detaillierte Beschreibung und Anleitung kann auf der Website von rapid7, den Entwicklern von *Metasploit*, vorgefunden werden [csb]. Zum derzeitigen Stand werden alle Hardware-Schnittstellen unterstützt, welche mit *SocketCAN* verwendet werden können. Die Modulentwicklung befindet sich noch in einer frühen Phase. Generell können jedoch viele bekannte Fahrzeug-Exploits als *Metasploit*-Modul implementiert und der Öffentlichkeit zur Verfügung gestellt werden.

### **CANToolz:**

Durch den modularen Aufbau und eine Web-Oberfläche bietet das pythonbasierte Framework *CANToolz* [asi] vielfältige Methoden zur Analyse von CAN-Traffic. Über MITM-Funktionen können Pakete mit bestimmten IDs blockiert werden. Mit einfachen Fuzzing-Funktionen kann das Bussystem automatisiert auf verwendete aber unbekannte IDs getestet werden. Pakete können nach der Speicherung wieder in den Bus übertragen werden und Statistiken können erstellt werden. Eine weitere nützliche Funktion besteht in der automatischen Erkennung von Security-Access-Paketen des UDS-Protokolls.

## **3.2 Aufbau von Testumgebungen**

Testumgebungen bieten die Möglichkeit, dass das Analysieren von Fahrzeugnetzwerken in einer kontrollierten Umgebung stattfinden kann. Pakete können ohne Auswirkungen auf reale Fahrzeuge und deren Systeme in Busnetzwerke versendet werden. Vor allem bei der Verwendung von Ansätzen des Fuzzy-Testings, bei welchen wiederholt zufällige Pakete generiert und deren bisher unbekannte Auswirkungen beobachtet werden, stellt dies die Wiederherstellung des ordnungsgemäßen Zustand von Testfahrzeugen sicher. Zu untersuchende Steuergeräte oder Komponenten können isoliert betrachtet werden, ohne dass eine Kommunikation mit anderen Fahrzeugbauteilen stattfinden kann.

### 3.3 Virtuelle Testumgebungen

Dieser Abschnitt beschreibt Testumgebungen, welche durch Rechner simuliert werden können. Derartige Umgebungen bieten den Vorteil, dass beispielsweise bei Demonstrationen keine kompletten Netzwerke vorgenötigt sind. Zusätzlich können Einstellungen vorgenommen werden, welche bei physischen Testumgebungen nicht oder nur mit unverhältnismäßigem Aufwand möglich wären. Dies umfasst unter anderem die Anpassung der verwendeten CAN-IDs, der Grad des Hintergrundrauschens und die Intervalle, in den für die Analyse relevante Datenpakete versandt werden.

Für linuxbasierte Betriebssysteme besteht die Möglichkeit der Nutzung von virtuellen CAN-Interfaces. Diese können nach der Erstellung von Programmen wie herkömmliche CAN-Interfaces zur Übertragung von Daten bzw. zur Traffic-Analyse genutzt werden.

Mithilfe folgender Kommandos kann ein derartiges virtuelles CAN-Interface mit dem Namen `vcan0` erstellt werden:

```
1 sudo modprobe can
2 sudo modprobe vcan
3 sudo ip link add dev vcan0 type vcan
4 sudo ip link set up vcan0
```

Listing 5: Software: Erstellung eines virtuellen CAN-Interfaces

**ICSim:** Mit dem Instrument Cluster Simulator (*ICSim*) [csa] ist ein Open-Source-Softwareprojekt verfügbar, mit welchem das Kombiinstrument von Fahrzeugen und dazugehörige CAN-Pakete simuliert werden können. Die Anwendung simuliert Tachometer, Blinkerstatus-Leuchten und eine Anzeige bezüglich des Öffnungsstatus der Türen inklusive den dazugehörigen CAN-Daten. Die Realisierung erfolgt mithilfe von virtuellen CAN-Schnittstellen.

Das Projekt besteht aus zwei Komponenten:

1. `icsim`: Anzeige des Kombiinstruments inklusive der momentanen Geschwindigkeit, Blinker- und Türstatus
2. `controls`: Kontrolle der virtuellen Umgebung und Generierung der CAN-Daten.

Mittels optionalen Parametern kann vor dem Start beider Komponenten der gewünschte Schwierigkeitsgrad angegeben werden. Dies beeinflusst die Zufälligkeit der Pakete und deren Inhalt. Der Seed, welcher hierbei nach dem Start von `icsim` ausgegeben wurde, muss als Parameter von `controls` angegeben werden, um die Synchronität beider Komponenten zu gewährleisten.

Nach dem Start beider Anwendungen erscheint folgendes virtuelle Kombiinstrument:

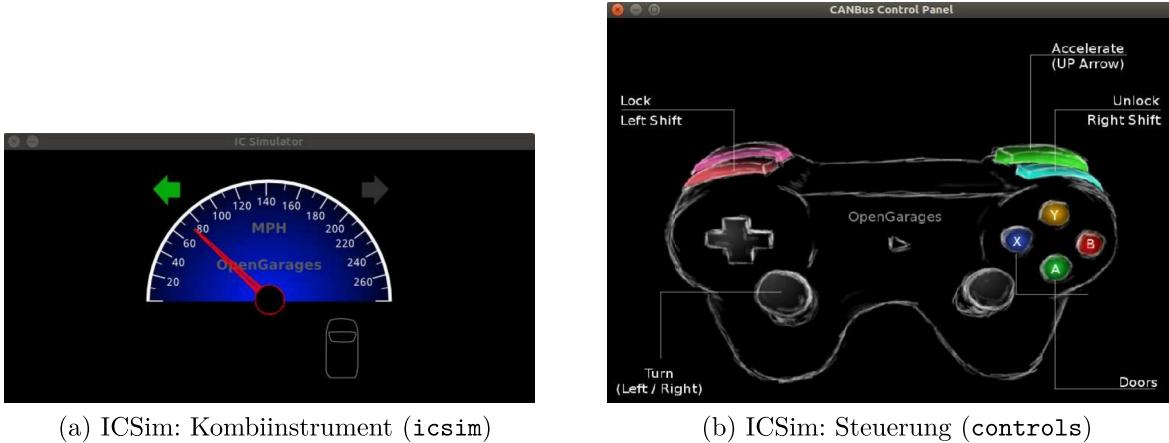


Abbildung 26: Software: ICSim

Die Geschwindigkeit des virtuellen Fahrzeugs kann mittels `controls` anhand der Pfeiltasten gesteuert werden. Neben der Anpassung der Tachometer-Nadel im `icsim`-Fenster erfolgt parallel die Generierung des entsprechenden CAN-Datenverkehrs auf dem virtuellen CAN-Interface. In Listing 4 kann ein Beispiel der generierten Daten eingesehen werden.

Das Herausforderung des virtuellen Netzwerkaufbaus ist das Ermitteln der CAN-IDs für:

- Geschwindigkeit inkl. valide Datenwerte
- Blinker-Status
- Tür-Status

Je nach Schwierigkeitsgrad, eingestelltem Hintergrundrauschen und Randomisierung kann die Verwendung einiger Analysewerkzeuge nötig sein, um alle IDs zu ermitteln. *ICSim* simuliert somit einen direkten Buszugriff und erlaubt das Testen von eigenen Analysemethoden und Anwendungen auf eine sichere Art und Weise.

#### UDSim:

Dieser Simulator [csc] zielt auf das UDS-Protokoll ab und implementiert drei Modi:

1. Learning: In diesem Modus wird der angeschlossene CAN-Bus auf UDS-Pakete überwacht. Die Anwendung erkennt UDS-fähige und kommunizierende Steuergeräte automatisch. Es ist kein vorheriges Wissen über das zu analysierende Fahrzeug nötig.
2. Simulation: UDSim ist in der Lage aus dem aufgezeichneten UDS-Traffic ohne eine Verbindung zum Fahrzeug eine bestehende Verbindung zu simulieren. Das vorherige Durchlaufen der Lern-Phase ist allerdings zwingend nötig, da das Werkzeug keinen UDS-Traffic *erfinden* kann. Jedoch ist es möglich, dass Antworten auf UDS-Anfragen gemäß der Spezifikation generiert werden können, selbst wenn keine entsprechenden Antworten aufgezeichnet wurden. Datenfelder werden hierzu bei Bedarf mit zufälligen Werten gefüllt.
3. Attack: Dieser Modus erlaubt das Fuzzing von UDS-Paketen, um beispielsweise versteckte Services zu ermitteln

## 3.4 Physische Testumgebungen

Diese Art von Testumgebungen besitzt den Anspruch, Fahrzeugbusse *möglichst* realitätsnah in einer kontrollierten Umgebung bereitzustellen. Hierdurch wird eine gewisse Übertragbarkeit der Laborergebnisse auf reale Fahrzeuge ermöglicht.

Es wird meist eine minimale Menge von Busknoten verwendet, um ein in sich geschlossenes Fahrzeugnetzwerk aufzubauen. Dies verhindert die Fremdeinwirkung von bisher nicht betrachteten Netzteilnehmern. Mit mehr eingebrachten Knoten steigt meist der Anwendungsbezug zu realen Fahrzeugen – die Komplexität des aufgebauten Labornetzwerks steigt jedoch gleichzeitig. Es bietet sich an, stets Netzwerkkomponenten aus demselben Ursprungsfahrzeug zu verwenden, um die Kompatibilität der Busteilnehmer zu gewährleisten.

Zum Betrieb der aufgebauten Testumgebung ist im Unterschied zu virtuellen Umgebungen oft proprietäres Wissen oder Reverse-Engineering nötig. Ohne Kenntnisse über die Pinbelegung kann zunächst keine Stromversorgung bzw. Busverbindung aufgebaut werden. An dieser Stelle können herstellerinterne Datenblätter hilfreich sein, sofern diese verfügbar sind. Alternativ können die Platinen der Netzwerkgeräte analysiert werden, um Pins zur Stromversorgung und Datenübertragung ausfindig zu machen.

Grundsätzlich werden folgende Materialien zum Aufbau eines beschriebenen Labornetzwerks benötigt:

- 12V Stromquelle für jeden Busteilnehmer
- Beliebige Anzahl kompatibler Netzwerknoten
- Datenleitungen mit Steckverbindern (*In eigenen Experimenten wurden GPIO-Kabel verwendet*)
- Hardware zur Businteraktion (z.B. USB-Adapter und D-Sub DB15 Breakout-Boards)

Im Anschluss wird der Aufbau eines minimalen Labornetzwerks mithilfe eines Kombi-instruments erläutert. Dem Kombiinstrument zugehörige Analyseergebnisse können [5.1](#) entnommen werden. Im verwiesenen Abschnitt wird darüber hinaus die Erhebung von Informationen bezüglich den Pinbelegungen erläutert, welche an dieser Stelle vorausgesetzt werden.

Nachfolgende Komponenten wurden zum Aufbau verwendet:

- Komerzi Labornetzgerät PS2002H
- USB2CAN (siehe [3.1.1](#))
- REDACTED
- D-Sub DB15 Breakout-Board zur vereinfachten Verkabelung (optional)

Die Verkabelung der Komponenten erfolgt gemäß den ermittelten Informationen aus proprietären Quellen oder eigenen Analysen:

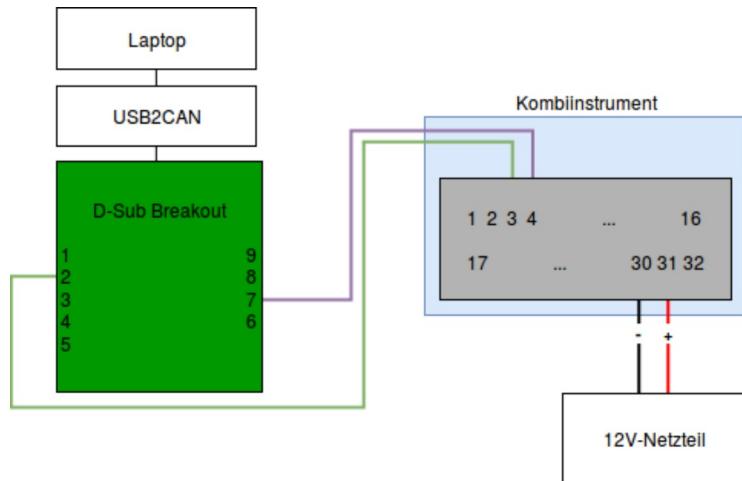


Abbildung 27: Analyse: Aufbau einer physischen Testumgebung

Die Pins des Kombiinstruments können mit Steckverbindern angeschlossen werden:

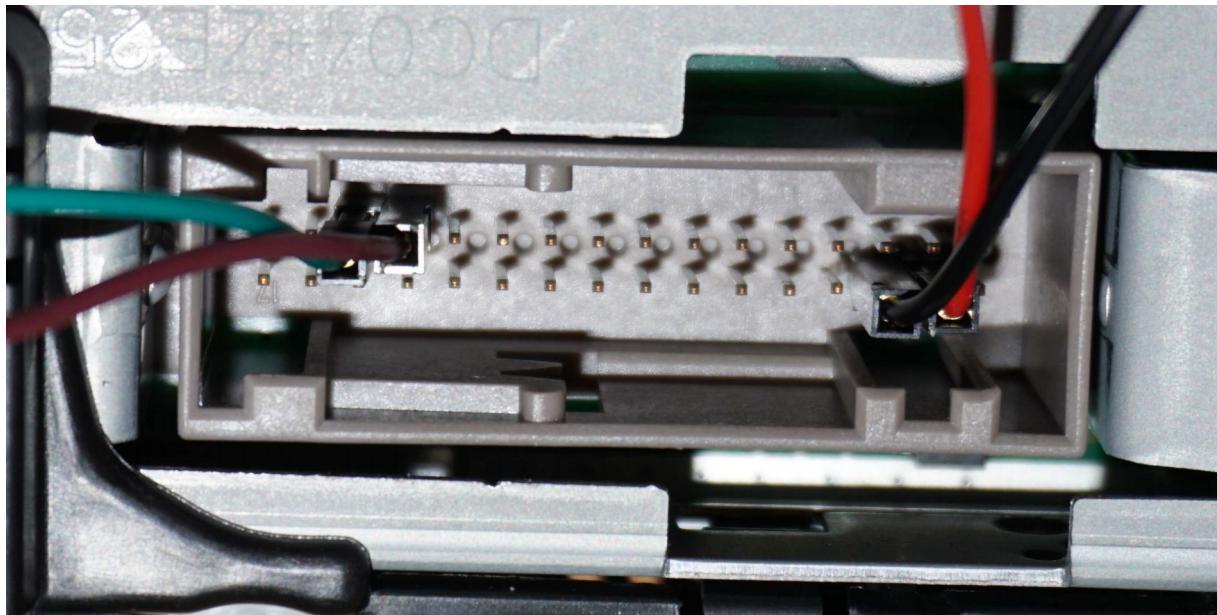


Abbildung 28: Analyse: Anschlusspins des Kombiinstruments

Im Anschluss liegt folgender Aufbau vor:

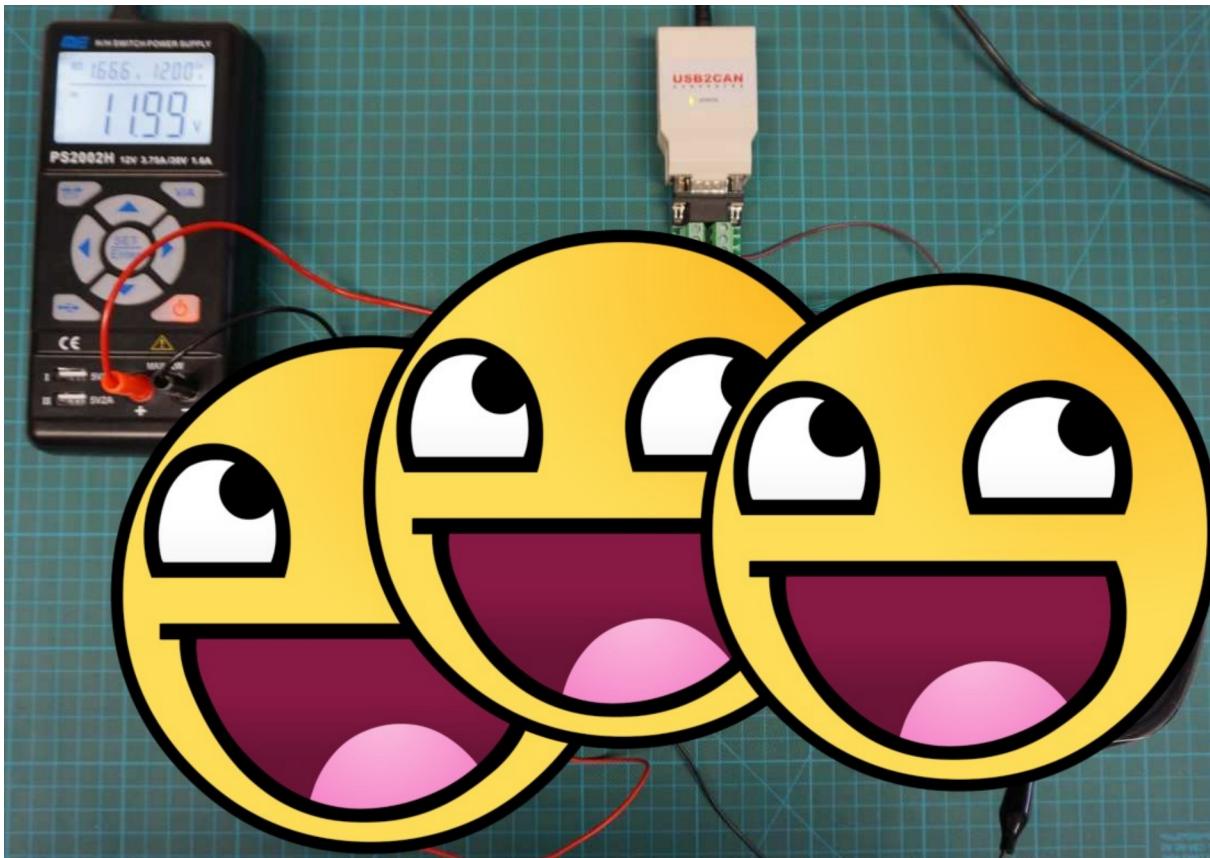


Abbildung 29: Analyse: Physische Testumgebung

Informationen zur Analyse von CAN-Knoten am Beispiel des abgebildeten Kombiinstruments befinden sich im Abschnitt 5.1.

### 3.5 Angriffsmethoden und Techniken des Reverse-Engineerings

Mithilfe dieses Abschnitts sollen einige Vorgehensweisen und Methoden des Reverse-Engineerings im Bezug auf Fahrzeugnetzwerke erläutert werden. Es soll eine Idee dafür entwickelt werden, welche Analysetechniken bei Fahrzeugen Anwendung finden können.

Zu beachten ist der bereits in Abschnitt 2.5 angesprochene Unterschied zwischen dem Buszugriff via OBD-II und über eine direkte Busverkabelung. Letzterer Ansatz ist aufwändiger, jedoch lassen sich oft nur über diese Zugriffsmethode schwerwiegende Befunde erheben. Nichtsdestotrotz sollten bei einer Analyse beide Methoden überprüft werden.

#### 3.5.1 Manuelle Requests

Oft müssen bei Analysen manuell Anfragen erstellt und deren Antwort beobachtet werden. Die Anwendungsgebiete dieser Vorgehensweise können vielfältig sein. Sie reichen von der Abfrage von Fahrzeugdaten mittels Standard-PIDs über das Fuzzing von potentiellen PIDs bis hin zur Interaktion mit proprietären Diensten. Generell kann diese Angriffsart auch als *Packet Injection* bezeichnet werden.

Wie manuelle CAN-Anfragen mittels `cansend` erstellt werden, ist anhand von folgendem Listing ersichtlich:

```
1 cansend can0 7DF#02010C
```

Listing 6: Reverse-Engineering: Manueller Request (Motordrehzahl)

Zur Demonstration der manuellen Paket-Versendung wurde im obigen Listing eine Diagnoseanfrage zur Ermittlung der aktuellen Motordrehzahl erstellt.

Nach der Angabe des CAN-Interfaces müssen CAN-ID und die dazugehörigen Nutzdaten inklusive Request-Modus und Paketlänge angegeben werden. Getrennt werden diese Daten durch eine Raute. Im obigen Beispiel werden folgende Angaben verwendet:

- 7DF: ID des ISO-TP-Broadcasts für 11-Bit-IDs [emoa]
- #: Trennzeichen
- 02: Länge der folgenden Daten
- 01: Modus des OBD-Requests: 0x01 steht für das Auslesen von aktuellen Daten
- 0C: OBD-PID zur Abfrage der aktuellen Motordrehzahl

Im erstellten Request wurde die Broadcast-ID des ISO-TP-Protokolls zur Ansteuerung des zuständigen Steuergeräts verwendet. Theoretisch könnte die Verwendung von ISO-TP als Transportprotokoll ausgelassen werden und mittels CAN direkt auf Busebene ein Diagnose-Request abgesetzt werden. Dies hätte jedoch den Nachteil, dass Antworten mit einer Länge von über 8 Bytes nur teilweise empfangen werden können [Smi16].

Als Antwort der Anfrage wird folgendes Paket empfangen:

```
1 can0 7E8 [8] 04 41 0C 1B 52 00 00 00
```

Listing 7: Reverse-Engineering: Diagnoseantwort (Motordrehzahl)

Jedes angesprochene Steuergerät antwortet auf eine Diagnoseanfrage mit einer Adresse von 0x7E8 beginnend. Jedes i-te antwortende Steuergerät antwortet somit mit der Adresse 0x7E8 + i. Die Antwort enthält 4 Nutzdaten-Bytes (0x04). Der Modus der Antwort beträgt den ursprünglichen Wert (0x01), welcher um 0x40 erhöht wurde, um die Anfrage positiv zu quittieren. Mit 0x0C wird signalisiert, dass sich der folgende Wert als Motordrehzahl interpretieren lässt.

Um aus dem Hexadezimalwert 0x1B52 den Dezimalwert der Motordrehzahl zu berechnen wird folgende allgemein gültige Formel angewandt [Wer14]:

$$W_{phys} = \frac{(W_{hex})_{10}}{2^n - 1} * (W_{max} - W_{min}) + W_{min}$$

Der Wert  $n$  repräsentiert in dieser Formel die maximale Bitanzahl des Wertes. Im Falle der Motordrehzahl sind dies 16 Bit. Dieser Wert kann mitsamt der Minimal- und Maximalwerte für jeden möglichen Standard-Request aus [Wer14] entnommen werden.

Im Falle der erhaltenen Antwort ergibt sich für die aktuelle Motordrehzahl folgender Wert:

$$W_{phys} = \frac{(1B52)_{10}}{2^{16} - 1} * (16383, 75 - 0) + 0 = 1748,5 \frac{\text{Umdrehungen}}{\text{Minute}}$$

Mittels des vorhergehenden Beispiels soll eine Idee für den generellen Kommunikationsprozess entwickelt werden. Für tiefgreifende Netzwerkanalysen sind Standard-PIDs meist nicht das vielversprechendste Ziel. Bei der Analyse auf unbekannte Netzwerk-Services bzw. PIDs muss dennoch ein vergleichbarer Prozess durchlaufen werden. Gewünschte Pakete werden mit Bezug zu dem verwendeten Protokoll erstellt, versandt und eine eventuelle Antwort abgewartet. Die Nutzdaten der Antwortpakete müssen anschließend interpretiert werden. Die Verwendung von Standard-PIDs erlaubt zudem erste Tests der Fahrzeugkommunikation.

### 3.5.2 Fuzzing

Diese Analysemethode verfolgt den Ansatz des randomisierten Testens. Mittels zufällig generierten Paketen sollen proprietäre Dienste aufgespürt oder bereits bekannte Services auf Fehlerverhalten oder bisher unbekannte Verhaltensweisen überprüft werden.

Neben den bereits erwähnten Standard-IDs der verschiedenen Protokolle kann jeder Fahrzeugherrsteller eigene IDs implementieren. Diese dienen beispielsweise zur Kommunikation unter den Steuergeräten oder der Interaktionen mit einem externen Diagnosegerät. Das Ziel von Fuzzing kann das Ermitteln bisher unbekannter valider IDs im Zusammenhang mit entsprechenden Nutzdaten sein.

Ein Werkzeug, mittels welchem CAN-Fuzzing betrieben werden kann, ist `cangen` aus dem `can-utils`-Paket. Hiermit können zufällige CAN-Pakete erzeugt und in den Bus versendet werden. Ein einfaches Beispiel ohne zusätzliche Parameter ist im folgenden Listing einzusehen:

```

1 $ cangen vcan0 &
2 $ candump vcan0
3   vcan0 379 [0]
4   vcan0 18B [8] CF 6E CD 20 A4 D4 EB 3E
5   vcan0 61B [8] 9B C7 81 07 95 C6 BB 2D
6   vcan0 136 [8] D6 99 0D 7C 93 51 BC 19
7   vcan0 374 [2] 46 23
8   vcan0 2DB [8] D9 8D C9 36 44 7A 26 2D
9   vcan0 743 [8] E6 03 4F 09 59 B9 6D 37
10  [...]
```

Listing 8: Reverse-Engineering: Fuzzing mit `cangen` (1)

Mit zusätzlichen Parametern können zum Beispiel 29-Bit-IDs inkrementell mit jeweils sieben zufälligen Datenbytes getestet werden:

```

1 $ cangen vcan0 -e -I i -L 7 &
2 $ candump vcan0
```

```

3   vcan0  00000000  [7]  8D 7D 2F 2A B3 98 AE
4   vcan0  00000001  [7]  37 A3 C0 63 BF 9F 97
5   vcan0  00000002  [7]  74 50 AD 19 97 0E B1
6   vcan0  00000003  [7]  75 B8 F6 2A 53 76 1F
7   vcan0  00000004  [7]  5D B1 55 2F 48 5E 96
8   vcan0  00000005  [7]  FA B5 C6 5A 53 A4 FA
9   vcan0  00000006  [7]  58 1C EE 3F 20 D7 2A
10  [...]

```

Listing 9: Reverse-Engineering: Fuzzing mit cangen (2)

Je nach Situation, Implementierung und verwendeter Software sind weitere Anpassung bezüglich der randomisierten Paketerzeugung möglich. Das Zeitintervall, innerhalb welchem neue Pakete in den Bus versendet werden, kann angepasst werden. Bei einer zu hohen Datentransferrate kann hierdurch eventuell ein Busnetzwerk kurzzeitig überlastet werden. Anschließend können manuell Datenwerte getestet werden, um den Wertebereich einzuschränken und gewünschte Aktionen auszuführen.

Generell sollte bei einem Fuzzing-Prozess stets ein Abzug (Dump) der generierten Pakete lokal gespeichert werden. Dies stellt die Reproduzierbarkeit für weitere Überprüfungen sicher, insofern eine bestimmte ausgelöste Aktion eines Fahrzeugnetzwerks beobachtet werden kann. Mittels Techniken des nächsten Abschnitts können aus großen Paket-Dumps relevante Pakete gefiltert werden, um bestimmte Aktionen auslösen.

### 3.5.3 Dump-Erstellung und Replay-Attacken

Um Daten von Fahrzeugnetzwerken aufzuzeichnen kann die Anwendung `candump` des `can-utils`-Pakets verwendet werden. Um Pakete direkt in das für Replay-Attacken benötigte Format zu speichern wird folgender Befehl verwendet:

```
1 candump -L <Interface> > <Dateiname>
```

Listing 10: Reverse-Engineering: Dump-Erstellung

Dies bewirkt das Aufzeichnen von Zeitstempeln, mittels welchen die exakte Wiedergabe der aufgezeichneten Datei möglich wird. Die Zeitabstände der aufgezeichneten Pakete werden somit bei Replay-Attacken beibehalten.

Mittels Wireshark ist darüber hinaus die Erstellung von Paket-Abzügen im `.pcap`-Format möglich.

Sofern die gewünschten Pakete durch `candump` gespeichert wurden, kann anschließend mit `canplayer` die Replay-Attacke ausgeführt werden:

```
1 canplayer -I <Dateiname> <Ausgangs-Interface>=<Ursprungs-Interface>
```

Listing 11: Reverse-Engineering: Replay-Attacke

Das ursprüngliche Interface entspricht in diesem Zusammenhang dem Interface, von welchem die Daten einst aufgezeichnet wurden. Das Ursprungs- und Ausgangs-Interface können sich unterscheiden, da beispielsweise von `vcan1` gespeicherte Pakete in einer späteren Sitzung mit dem Interface `vcan0` übertragen werden müssen.

Bei der Arbeit mit großen Paketmengen kann oft folgende Situation auftreten: Es liegt eine Dump-Datei mit einer Vielzahl von Paketen vor. Es ist bekannt, dass in der Datei ein Paket enthalten ist, welches bei einem Fahrzeug eine bestimmte Aktion auslöst. Eine effiziente Lösung um das relevante Paket zu bestimmen ist an das binäre Suchverfahren angelehnt, bei welchem der Suchraum nach jedem Test halbiert wird. Folgende Grafik veranschaulicht den Suchvorgang:

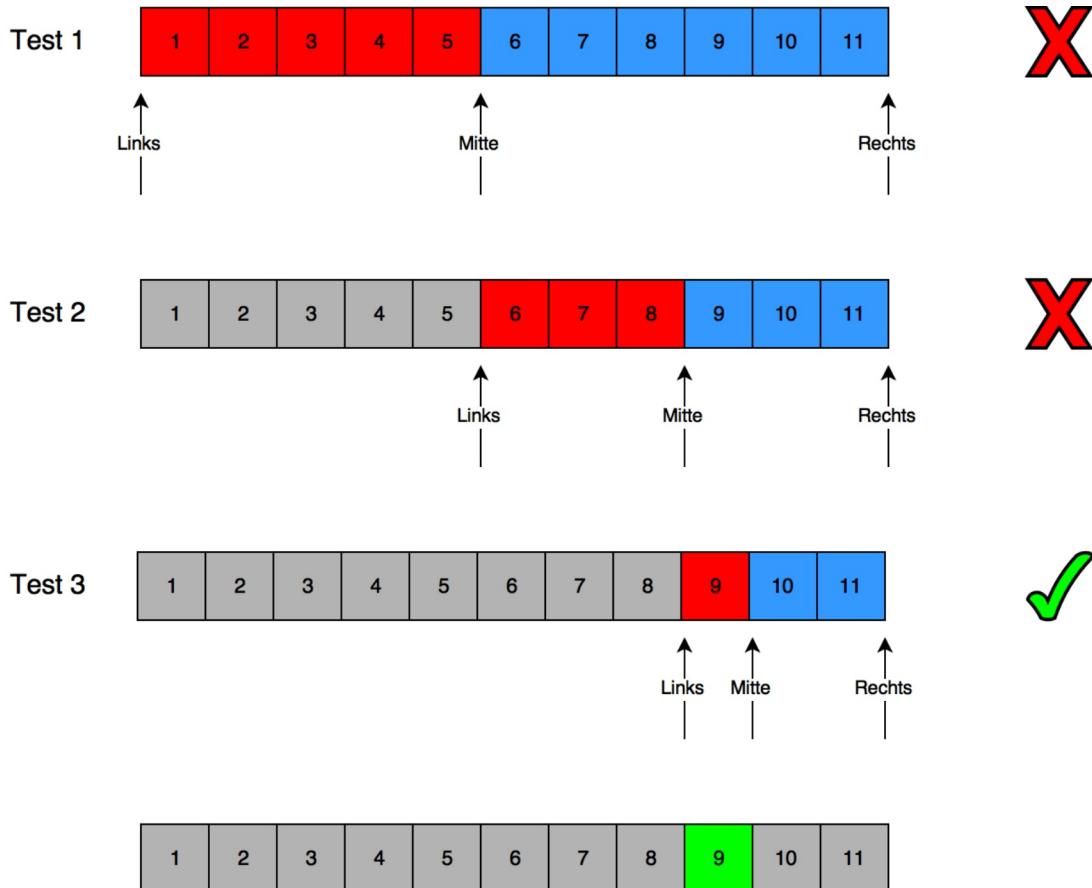


Abbildung 30: Reverse-Engineering: Binäre Paketsuche

Vor jedem Test wird der momentane Suchraum halbiert. Sofern der Test im halbierten Suchraum erfolgreich ist, werden die aktuell betrachteten Pakete für den nächsten Durchlauf verwendet (rot). Ansonsten bilden die restlichen Pakete (blau) den neuen Suchraum. Dieser Vorgang wird solange wiederholt bis das gesuchte Paket bestimmt wurde. Es ist möglich, dass zwei oder mehrere Pakete zusammen eine Reaktion des Fahrzeugs auslösen. In diesem Fall kann der durch den Suchvorgang erzeugte Suchbaum von unten nach oben durchlaufen werden, um eine minimale Menge von relevanten Paketen zu bestimmen. Die Verwendung von randomisierten Suchverfahren stellt eine Alternative hierzu dar, welche im Rahmen der Analysesoftware-Entwicklung implementiert wurde (4.2.3).

### 3.5.4 Angriffe auf UDS

Dieser Abschnitt widmet sich den Angriffsmöglichkeiten auf das UDS-Protokoll. Der Fokus derartiger Attacken liegt meist im Erlangen eines privilegierten Diagnosezugangs zu

den internen Fahrzeugsystemen. Durch diesen Zugriff sind nahezu beliebige Manipulationen von Steuergeräten und deren Konfiguration möglich. Anderweitige Ziele können das Verwenden oder Ausnutzen von proprietären Diagnosediensten sein.

### Ermittlung von Diagnosediensten

Mit der Software `caringcaribou` [cbu] können standardisierte und proprietäre Dienste innerhalb des Netzwerks aufgespürt werden. Um dies zu erreichen wird eine zweistufige Suche durchgeführt:

1. **dcm discovery:** An jede durch UDS verwendbare CAN-ID wird eine Tester-Present-Nachricht versandt (vgl. 2.4). Dies hat zur Folge, dass für jede Service-ID im Bereich 0x000 bis 0x7FF eine Verbindungsanfrage versandt wird. Sofern ein Dienst auf einer entsprechenden ID vorhanden ist, erfolgt eine Antwort auf das versandte Tester-Present-Paket. Antworten entsprechen entweder Paketen mit ID-Werten von 0x40 + Service-ID (positive Antwort) oder 0x7F (Fehlerantwort). In beiden Fällen liegt ein Indiz vor, dass auf der versendeten ID ein Diagnosedienst erreichbar ist. Fehlende Antworten bedeuteten dagegen, dass unter der angegebenen ID kein Dienst vorhanden ist.
2. **dcm services:** Nachdem die CAN-ID des Dienstes ermittelt wurde, werden im folgenden Schritt die unterstützten Dienstfunktionen ermittelt. Hierbei werden für die im vorigen Schritt ermittelte CAN-ID bzw. Dienst-ID alle möglichen UDS-SIDs durch eine Brute-Force-Attacke getestet. Antworten ungleich 0xFF711 implizieren die Unterstützung der angefragten UDS-Funktion [cbu]. Eine positive Antwort auf die Anfrage mit der UDS-SID 0x23 deutet somit beispielsweise auf die Unterstützung der Funktion *Read Memory By Address* hin. Zu beachten ist jedoch, dass zur Interaktion mit zahlreichen Diensten aufgrund fehlender Privilegien der Standard-Diagnosesitzung eine Rechteausweitung durchgeführt werden muss.

### Reverse-Engineering des Authentifizierungsalgorithmus

Implementierungen von Challenge-Response-Verfahren können eine sehr unterschiedliche Komplexität aufweisen. Es ist möglich, dass der Seed zur Authentifizierung lediglich mit einem statischen Wert addiert werden muss, um vom Fahrzeug als korrekt angesehen zu werden. Vergleichend dazu sind jedoch auch komplexe Algorithmen möglich, welche nicht oder nur extrem schwer durch schlichtes Beobachten mehrerer Testwerte ermittelt werden können.

Um den Austausch von Seed- und Schlüsselwerten zwischen Fahrzeug und Software bzw. Testgerät aufzuzeichnen, eignet sich ein Y-Kabel (siehe 3.1.1). Mit der Verwendung dieses Kabeltyps wird die Verbindung zwischen Fahrzeug und Testgerät ermöglicht, während der Datenverkehr simultan aufgezeichnet werden kann. Bei schwachen Authentifizierungsverfahren ist im Anschluss ein Reverse-Engineering anhand der ermittelten Datenpunkte in Form von Seed-Schlüssel-Paaren möglich.

Das Wissen über derartige Algorithmen ist sehr begehrt. Es existiert ein Schwarzmarkt, auf welchem Algorithmen und gültige Schlüssel ausgetauscht werden:

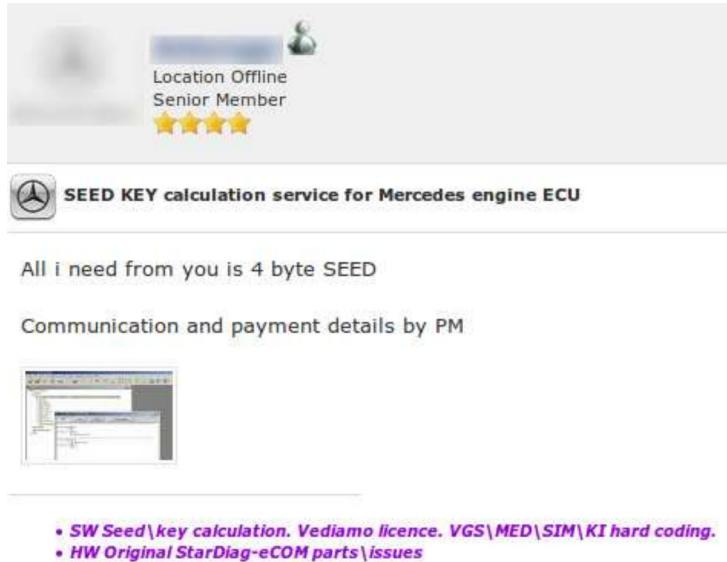


Abbildung 31: Analyse von Fahrzeugnetzwerken: Forenbeitrag eines Schlüsselverkäufers

Im obigen Foren-Posting verkauft ein *Händler* gültige UDS-Schlüssel, sofern der Käufer den vom Fahrzeug verwendeten Seed bereitstellt. Der Seed ändert sich *meist* bei jedem Verbindungsauflauf. Dies bedeutet, dass die erworbenen Schlüssel nur einen temporären privilegierten Zugang ermöglichen. Für den Erwerb der verwendeten Algorithmen müssen dementsprechend mehr finanzielle Mittel aufgebracht werden. Meist werden diese Algorithmen jedoch nicht verkauft um das Geschäftsmodell aufrecht zu erhalten.

Eine Alternative, über welche auch stärkere Authentifizierungsalgorithmen angegriffen werden können, ist das [Reverse-Engineering von Hersteller-Software](#).

### **Brute-Force-Attacken**

In seltenen Fällen kann das Testen aller möglichen Schlüsselwerte zum Erfolg führen. Dies ist der Fall, wenn stets derselbe Seed vom Fahrzeug versendet wird *und* kein softwareseitiger Schutz gegen derartige Attacken implementiert wurde. Dieses Szenario ist heutzutage extrem selten anzutreffen aber kann jedoch nicht gänzlich ausgeschlossen werden (siehe 5.3.5).

### **Replay-Attacken**

Insofern von einem Fahrzeugsteuergerät bei allen Authentifizierungsversuchen ein identischer Seed versendet wird und ein offizielles Testgerät des Herstellers bzw. eine Hersteller-Software verfügbar ist, lassen sich Replay-Attacken auf die Authentifizierung durchführen. Dabei muss lediglich der Datenverkehr einer erfolgreichen Authentifizierung, beispielsweise mithilfe eines Y-Kabels, mitgeschnitten werden. Da der vom Fahrzeug erhaltene Seed stets gleich ist, kann das aufgezeichnete Authentifizierungspaket für alle weiteren Verbindungsversuche genutzt werden.

### **Aufrechterhaltung von UDS-Sitzungen**

Wie bereits in 2.4 beschrieben, muss zur Aufrechterhaltung einer Diagnosesitzung das Tester-Present-Paket periodisch an das Fahrzeug versendet werden. Geschieht dies nicht, erlischt die Sicherheitsfreigabe und die privilegierte Sitzung wird automatisch geschlossen. Mittels folgenden Bash-Skript (angelehnt an [Smi16]) wird die permanente Aufrechterhaltung der Sitzung gewährleistet:

```
1 while true; do cansend <CAN-Iface> 7DF#013E; sleep 1; done
```

Listing 12: Analyse von Fahrzeugnetzwerken: UDS-Heartbeat

Die funktionale Adresse 0x7DF entspricht der UDS-Broadcast-Adresse, während 0x013E der Tester-Present-Nachricht entspricht.

### 3.5.5 Man-In-The-Middle-Angriffe

Klassische Netzwerkattacken, wie der Man-In-The-Middle-Angriff, können teilweise auf Fahrzeugnetzwerken übertragen werden. Hierbei kann ein entsprechend positionierten Angreifer den Datenverkehr zwischen Quelle und Ziel aufzeichnen und manipulieren. Ein bereits vorgestelltes Werkzeug, mit welchem derartige Vorgehensweisen ermöglicht werden, ist der *CANBadger* [jhc].

Hierfür können für beliebige Paket-IDs Regeln angelegt werden. Diese Regeln definieren die Art und Weise der Paketmanipulation. Die zu manipulierenden Pakete werden mit den im selben Schritt angelegten Masken beim Angriff logisch miteinander verknüpft. Hierdurch wird eine zeiteffiziente Veränderung von Paketen ermöglicht. Optional können zudem mathematische Operationen durchgeführt und die Manipulation an eine bestimmte Bedingung geknüpft werden.

### 3.5.6 Reverse-Engineering von Hersteller-Software

Diagnose- und Werkstattsoftware von Automobilherstellern und Drittanbietern bietet oft die Möglichkeit, dass Steuergeräte-Software aktualisiert und sensible Einstellungen angepasst werden können. Meist findet hierzu eine Authentifizierung über das KWP2000- oder UDS-Protokoll statt.

Sofern durch herkömmliche **Angriffe auf UDS** keine Ergebnisse erzielt werden können oder eine andere Vorgehensweise gewählt werden soll, kann die Analyse der verwendeten Werkstattsoftware eine vielversprechende Vorgehensweise sein. Die Sicherheitsforscher Charlie Miller und Chris Valasek konnten bereits zeigen, dass proprietäre Zugangsschlüssel und Authentifizierungsalgorithmen aus entsprechenden Anwendungen extrahiert werden können [miva].

Oft sind herstellereigene Softwarewerkzeuge nicht öffentlich verfügbar. Sofern diese nicht akquiriert werden können, bieten Softwarelösungen von Drittanbietern eine nützliche Alternative. Zu beachten ist jedoch, dass die Analyse von Software mittels Techniken des Reverse-Engineerings, wie zum Beispiel die Dekompilierung, illegal sein kann.

Ein beliebtes Werkzeug zur Durchführung des Reverse-Engineerings von Software ist *IDA*. In Kombination mit dem *Hex-Rays Decompiler* [hxr] kann eine Dekompilierung durchgeführt werden, um Pseudocode zu erzeugen. Dies bietet eine verbesserte Verständlichkeit des erzeugten Programmcodes. Wichtige Vorgehensweisen im Zusammenhang mit der Analyse durch *IDA* und proprietärer Software sind:

- Extrahierung von enthaltenen Strings: Oft sind Debugging-Kommentare oder Ausgabeanweisungen im Code enthalten. Anhand dieser Codesegmente ist eine Orientierung innerhalb der analysierten Binärdatei möglich. Zudem erklären Ausgabeanweisungen oft den vorhergehenden Code. Die Suche nach interessanten Strings kann, je nach Kontext, durch Suchanfragen beschleunigt werden.
- Untersuchung von eingebundenen Bibliotheken und Dateien: Diese Ressourcen enthalten oft für die Untersuchung relevante Methoden, Hinweise auf Schwachstellen oder sogar Zugangsschlüssel.
- Generierung von Pseudocode und Flussdiagrammen: Diese Methoden vereinfachen das Verständnis von Methoden bzw. großen Codesegmenten drastisch.
- Durch die dynamische Analyse mit dem in *IDA* integrierten Debugger kann die Anwendung während der Ausführung beobachtet werden. Dies bietet vor allem bei der Untersuchung von proprietären Authentifizierungsalgorithmen Vorteile, da Zwischenergebnisse sichtbar werden und theoretisch alle gewünschten Methoden ausgeführt werden können. Hierzu ist jedoch ein entsprechender Versuchsaufbau in Kombination mit den für die Authentifizierung notwendigen Steuergeräte notwendig. Alternativ kann dieser Versuchsaufbau gegenüber der untersuchten Software simuliert werden.
- Das Verändern von Routinen der untersuchten Software (*Patching*) kann ähnlich einem Debugging-Vorgang Zwischenergebnisse ausgeben, Anti-Debugging-Methoden entfernen oder sonstige Modifikationen in Hinsicht auf die Offenlegung von Algorithmen und Schlüsseln erlauben.

Die genannte List ist jedoch keinesfalls abschließend – oft führen kreative oder fortgeschrittene Methoden zum gewünschten Ziel. In Fällen in denen die Datenanalyse keine Ergebnisse erzielt, bietet das Reverse-Engineering von Herstellersoftware unter Umständen die letzte verbleibende Option, da das Extrahieren und das Analysieren der Steuergeräte-Firmware des Fahrzeugs viele zusätzliche Ressourcen erfordert.

# 5 Analyseergebnisse

## 5.1 Reverse-Engineering eines Kombiinstruments

Bei der Analyse von proprietären Protokollen und Komponenten handelt es sich meist um einen Blackbox-Test. Das bedeutet, dass keinerlei herstellerspezifische Informationen vorliegen. Dies erschwert zwar die Analysen, jedoch kann hierdurch auch der allgemeine Sicherheitsstand aus Sicht eines potentiellen Angreifers aufgezeigt werden. Darüber hinaus können Verfahren, Technologien und proprietäre Informationen gewonnen und offengelegt werden.

Aus diesen Gründen bietet sich eine Analyse eines CAN-Geräts am Beispiel eines Kombiinstruments an. Dieser CAN-Knoten kann isoliert betrieben und analysiert werden, da keine notwendigen Abhängigkeiten zu anderen Netzwerkkomponenten bestehen. Die REDACTED

durchgeführt. Der verwendete Laboraufbau wurde in Abschnitt 3.4 näher erläutert.

### 5.1.1 Reverse-Engineering der Pinbelegung

Zunächst musste die Stromversorgung für das Kombiinstrument hergestellt werden. Hierfür existiert ein Stecker mit 32 Pins auf der Rückseite des Geräts. Um eventuellen Schäden durch unsachgemäße Stromzufuhr auf nicht dafür ausgelegten Pins vorzubeugen, wurde die Platine des Kombiinstruments analysiert. Die Rückseite der Platine ist auf folgender Abbildung zu sehen:

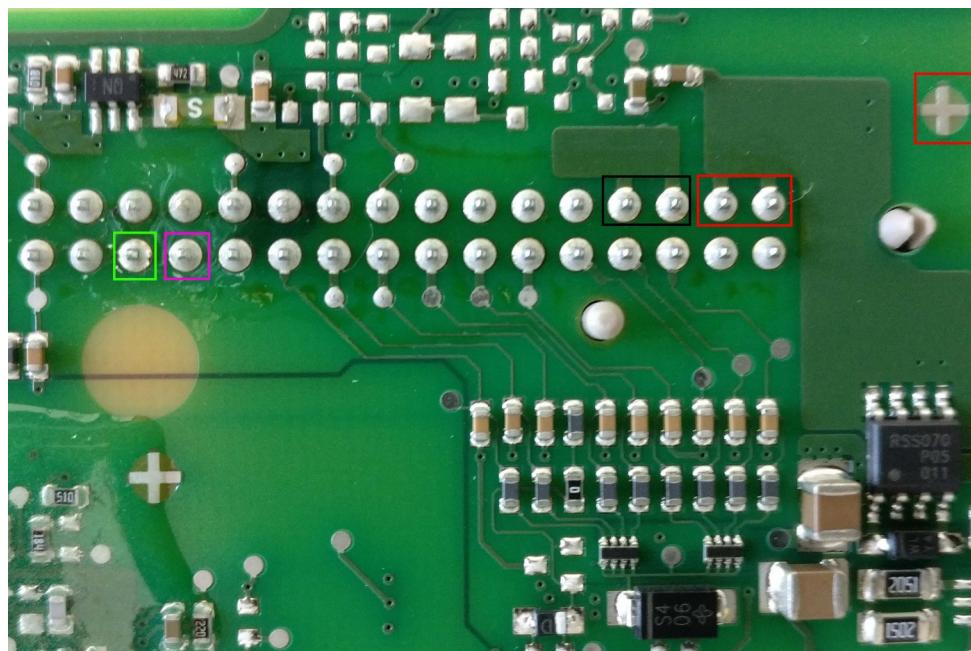


Abbildung 36: Ergebnisse Kombiinstrument: Reverse-Engineering der Pinbelegung

Es ist zu erkennen, dass zwei Pins miteinander verbunden und mit einem +-Symbol gekennzeichnet wurden (siehe rote Markierung). Dies legt nahe, dass es sich dabei um die

für die 12V-Stromzufuhr relevanten Pins handelt. Links neben der roten Markierung befinden sich zwei weitere miteinander verbundene Pins, auf welchen Masse-Pins vermutet werden. Die Stromzufuhr auf 12V- und Masse-Pins bestätigte die Vermutungen.

Die Identifizierung der Pins für CAN-High und CAN-Low kann sich je nach Gerät ohne Stromlaufplan als schwierig erweisen:

- Die Leiterbahnen für Daten-Pins auf der Platine sind im Vergleich zu Stromleitungen weniger offensichtlich erkennbar
- Oft werden Bauelemente auf Platinen unkenntlich gemacht oder durch spezielle Beschichtungen vor Analysen geschützt. Eine einfache Verfolgung von Leiterbahnen zu einem CAN-Transceiver kann somit nicht immer durchgeführt werden.

Um dennoch die relevanten Pins ausfindig zu machen gibt es generell zwei Möglichkeiten:

- Testen aller Pin-Kombinationen für CAN-High und CAN-Low
- Zugriff auf interne Dokumente von Automobilherstellern durch im Internet widerrechtlich veröffentlichte Diagnosesoftware.

Im Falle des analysierten Kombiinstruments konnten die Daten-Pins für CAN-High (grün) und CAN-Low (violett) durch das Testen einiger Pin-Kombinationen wie auf obiger Abbildung zu erkennen identifiziert werden. Um das Ergebnis zu verifizieren, wurden zufällig Pakete mittels `cangen` versendet und Bewegungen der Tachometer-Zeiger seitens des Geräts beobachtet.

Kombiinstrumente sind nach einer erfolgreichen Stromzufuhr meist noch nicht vollständig aktiv. Es muss zunächst ein Zündungssignal empfangen werden, bevor alle Funktionen zur Verfügung stehen. Dies kann, je nach Gerät, auf zwei Arten geschehen:

1. Besonderer Strom-Pin für Zündungs-Plus
2. Aktivierung der Zündung über ein CAN-Paket

Da die beiden einzigen auf dem Kombiinstrument vorhandenen 12V-Pins miteinander verbunden sind und durch Fuzzing eine Bewegung der Zeiger ausgelöst werden kann, wird die zweite Option als zutreffend vermutet. Es folgt somit zunächst die Identifizierung des CAN-Zündungspakets.

### 5.1.2 Reverse-Engineering der CAN-Matrix

Um Pakete oder Paketmengen, welche eine bestimmte Aktion auslösen, zu bestimmen kann generell folgendes Schema angewandt werden:

1. Fuzzing inklusive Aufzeichnung der zufällig erzeugten Daten
2. Beobachtung der Reaktionen des CAN-Geräts

3. Abbruch des Fuzzings bei der Beobachtung der gewünschten Reaktion
4. Ausführen der binären Paketsuche (siehe 4.2.3), um eine minimale Menge von relevanten Paketen zu bestimmen

Auf diese Art und Weise konnte das Zündungs-Paket identifiziert werden. Dieses Paket muss in einem Intervall von 100ms versandt werden, da sonst die Deaktivierung des Kombiinstruments erfolgt.

In seltenen Fällen sind mehrere Pakete für eine einzelne Aktion relevant. In diesen Fällen kann das Paket-Timing entscheidend sein. Hierauf ist bei der Ausführung der Paketsuche zu achten. Bei einer manuellen Suche können beispielsweise alle potentiellen Pakete schnellstmöglich in einer Endlosschleife versandt werden, um den Faktor des Timings zu minimieren.

Ist eine minimale Paketmenge identifiziert, ist es meistens sinnvoll die in den Paketen enthaltenen Nutzdaten zu analysieren. Wurde zum Beispiel das Paket identifiziert, welches den linken Blinker aktiviert kann mit etwas Analysearbeit eventuell aus diesen Daten das Paket für die Aktivierung des rechten Blinkers bestimmt werden. Im Falle der Geschwindigkeitsanzeige mussten vier relevante Bytes identifiziert und auf ihre unterschiedlichen Werte im Bezug auf die eingestellte Geschwindigkeit untersucht werden. Im Anschluss konnte ein Python-Skript entwickelt werden, welches aus der gewünschten Geschwindigkeit die CAN-Nutzdaten für das Geschwindigkeits-Paket berechnet und ausgibt. Dieses Skript ist zusammen mit der gesamten ermittelten CAN-Matrix des Kombiinstruments (8) im Anhang enthalten.

Beim Fuzzing von CAN-Geräten sollte generell mit Vorsicht vorgegangen werden. Während Laborumgebungen meist weniger kritisch sind, können in Fahrzeugen unerwünschte und oft nicht direkt ersichtliche Nebeneffekte durch zufällige Pakete auftreten. Durch einen Fuzzing-Vorgang mit maximaler Geschwindigkeit wurde die Wegfahrsperrre des Kombiinstruments aktiviert, welche auch durch weiteres Fuzzing nicht wieder deaktiviert werden konnte. Dies ist auf der Abbildung 29 anhand des Schriftzugs **SAFE** zu erkennen. Ein weiteres Beispiel stellt die versehentliche Deaktivierung des gesamten Bremssystems eines Fahrzeugs durch die Sicherheitsforscher Miller und Valasek dar [miva].

### 5.1.3 Interaktion mit Diagnosediensten

Mithilfe des Analysewerkzeugs **caringcaribou** (vgl. 3.5.4) konnten diverse Diagnosedienste des Kombiinstruments ausfindig gemacht werden. Es konnten hierbei zwei CAN-IDs identifiziert werden, unter welchen zahlreiche UDS-Dienste angesprochen werden können. Eine Übersicht der unterstützten UDS-SIDs bietet Tabelle 10.

Die beschriebenen UDS-Dienste können über CAN-Pakete im ISO-TP-Paketformat angesprochen werden, da jenes Paketformat von UDS unterstützt wird (vgl. 2.3.2):

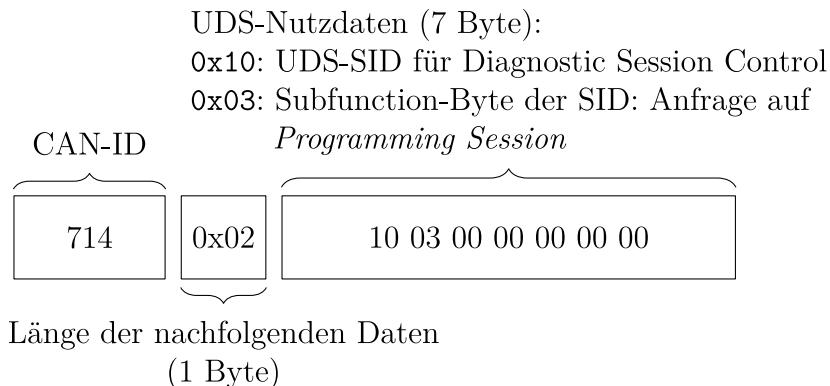


Abbildung 37: Ergebnisse Kombiinstrument: UDS-Anfrage in Verbindung mit CAN

Das obige UDS-Paket spricht den UDS-Dienst mit dem Namen *Diagnostic Session Control* und SID 0x10 auf der CAN-ID 714 an. Als Parameter wird 0x03 verwendet – das Anfordern einer Programmiersitzung auf dem Kombiinstrument. Diese Nachricht wird mit folgender Antwort quittiert:

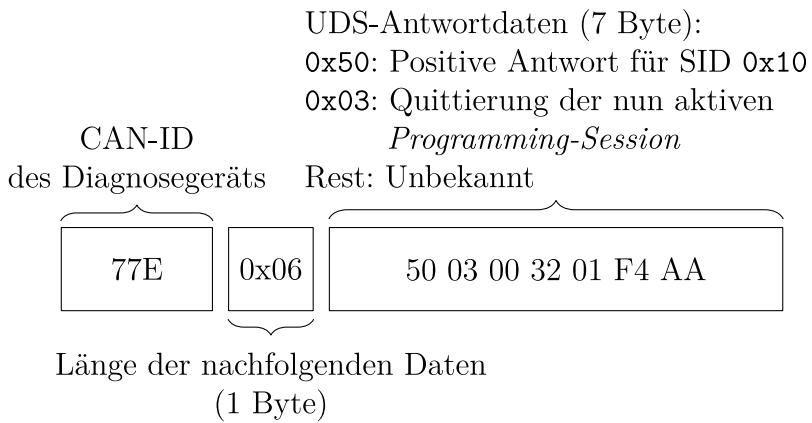


Abbildung 38: Ergebnisse Kombiinstrument: UDS-Antwort des Kombiinstruments

Durch die Antwort des Kombiinstruments wurde die Anfrage seitens des angesprochenen Dienstes positiv beantwortet. Dies ist anhand des ersten Bytes der Nutzdaten zu erkennen: Bei positiven Antworten wird an dieser Stelle der Wert des SID-Bytes der ursprünglichen Anfrage plus 0x40 verwendet. Bei negativen Antworten ist der Wert 0x7F enthalten. Neben der im Beispiel angeforderten Sitzung können weitere proprietäre Sitzungs-Modi durch Fuzzing des Bytes, welches für den angeforderten Sitzungs-Modus verantwortlich ist, aktiviert werden. Die Funktionen der einzelnen Sitzungen sind jedoch unbekannt.

Bevor die durch entsprechende Sitzungs-Modi freigeschalteten Operationen ausführt werden können, muss oft eine Authentifizierung durchlaufen werden. Beispielsweise sind nach Anforderung einer Programmier-Sitzung die Operationen für das Lesen und Schreiben des Speichers freigeschaltet. Ohne eine vorherige Authentifizierung werden diese Anfragen vom Steuergerät jedoch meist verworfen oder gefiltert. Das bedeutet, dass je nach angeforderter Speicherregion eine Authentifizierung erforderlich sein kann.

Aus diesem Grund existiert der Dienst *Security Access*, welcher mittels der SID 0x27 erreicht werden kann. Hiermit kann die Challenge-Response-Authentifizierung gegenüber dem Steuergerät erfolgen, um einen erweiterten Zugriff zu erhalten. Die Anforderung eines Seeds konnte mit dem analysierten Gerät jedoch nicht gemäß UDS-Spezifikation durchgeführt werden, da lediglich ein generischer Fehlercode als Antwort des Authentifizierungs-Dienstes empfangen wird. Möglicherweise ist die Authentifizierung an weitere Nebenbedingungen gekoppelt, welche im isolierten Betrieb des Kombiinstruments nicht erfüllt werden können.

Vereinzelt können Werte aus dem Speicher des Geräts durch den Dienst *Read Memory By Identifier* ohne vorhergehende Authentifizierung ausgelesen werden. Es scheint sich um statische Werte zu handeln, von welchen die Bedeutung bislang nicht identifiziert werden konnte.

Um eine weitergehende Analyse zu ermöglichen und um mit sicherheitskritischen Diensten wie *Write Data By Identifier* oder *Request Download* zu interagieren, müsste zunächst das Authentifizierungs-Problem gelöst werden. Dies ist ohne zusätzliche proprietäre Informationen höchstwahrscheinlich nur mit erheblichem Zeitaufwand zu bewältigen, sodass die Analyse der UDS-Dienste an dieser Stelle beendet wurde.

## 5.2 Reverse-Engineering von Wartungssoftware

### 5.2.1 REDACTED

Diese herstellerspezifische Anwendung des Automobilbauers X implementiert Methoden bezüglich Authentifizierungs- und Programmievorgängen. Das Ford-Pendant dieser **Integrated Diagnostic Software (IDS)**-Anwendung wurde bereits teilweise analysiert und offengelegt [miva]. Auf den ersten Blick sind die grafischen Oberflächen und der Codeaufbau von Ford -und X-Software vergleichbar. Im Rahmen der Analyse soll geklärt werden, ob Unterschiede vorliegen und welche Algorithmen zur Authentifizierung implementiert wurden.

Die vermutlich zur Authentifizierung verwendeten Schlüssel sind wie im Falle der Ford-Software [miva] am Anfang der Datei `AlgData.dll` als Array aufgelistet:

.rdata:10006724	db	52h	:	R
.rdata:10006725	db	45h	:	E
.rdata:10006726	db	40h	:	M
.rdata:10006727	db	41h	:	A
.rdata:10006728	db	54h	:	T
.rdata:10006729	db	0		
.rdata:1000672A	db	0		
.rdata:1000672B	db	0		
.rdata:1000672C	db	5		
.rdata:1000672D	db	0		
.rdata:1000672E	db	0		
.rdata:1000672F	db	0		
.rdata:10006730	db	54h	:	T
.rdata:10006731	db	41h	:	A
.rdata:10006732	db	40h	:	M
.rdata:10006733	db	45h	:	E
.rdata:10006734	db	52h	:	R

Abbildung 39: Ergebnisse IDA: Teil des Schlüssel-Arrays der Diagnosesoftware

Interessanterweise sind die Schlüssel mit denen der Ford-Software identisch. Das entsprechende Array kann über die String-Suche aufgefunden werden und enthält unter anderem zahlreiche Schlüssel, welche verdächtige ASCII-Repräsentationen besitzen:

- XXX
- Jesus
- XXX
- PANDA
- TAMER

Die vollständige Liste aller extrahierten Schlüssel ist im Anhang dieser Arbeit vorzufinden (8).

Es besteht zudem die Möglichkeit, dass die enthaltenen Schlüssel verschiedene Rechte des Zugriffs erlauben. Dies würde mit den Ergebnissen von Miller und Valasek übereinstimmen. Unterschiede bestehen höchstwahrscheinlich auch darin, für welche Steuergeräte welche Schlüssel verwendet werden können.

Ein Algorithmus, welcher vermutlich zur Authentifizierung gegenüber Suzuki-Fahrzeugen verwendet werden kann, konnte extrahiert werden. Folgendes Codesegment, in welchem der Algorithmus verwendet wird, ist das Ergebnis einer Suche nach dem Stichwort *Key*:

```

v116 = (int *)sub_1008DC60((__int16)v115, v121, v122, v124, v119, v129, v126);
v49 = v116;
sprintf(&v111, "*** - return iKey=0x%x", v116);

```

Abbildung 40: Ergebnisse IDA: Debug-Statement und Aufruf des Schlüsselalgorithmus

Überraschend ist hierbei das Vorkommen des Namens vom Fahrzeughersteller XX im abgebildeten Debug-Statement. Offensichtlich wird die Methode `sub_1008DC60` zur Schlüsselberechnung verwendet. Dieser Algorithmus ist folgendermaßen in Form von Pseudocode aufgebaut:

```

1 for ( i = 0; i < 3; ++i )
2 {
3     v17 = ~(*(&key + 2 * i) ^ seed_6);
4     v18 = v17 >> 7;
5     v17 *= 2;
6     v17 += v18;
7     v7 = HIBYTE(seed_6) ^ v17;
8     v17 = ~(*(&seed_1 + 2 * i) ^ HIBYTE(seed_6) ^ v17);
9     v18 = v17 >> 4;
10    v17 *= 16;
11    v17 += v18;
12    HIBYTE(seed_6) = seed_6 ^ v17;
13    LOBYTE(seed_6) = v7;
14 }
15 return (unsigned __int8)seed_6 |
16     (unsigned __int16)((HIBYTE(seed_6) << 8) & 0xFF00);

```

Der Inhalt der verwendeten Variablen kann über den Aufruf von `sub_1008DC60` nachverfolgt werden. Die Variable `key` entspricht im obigen Pseudocode dem ECU-Schlüssel. Die verbleibenden Parameter `seed_x` bilden gemeinsam den Seed, welcher die Challenge des Fahrzeugs darstellt. Interessanterweise werden lediglich Seed-Byte 1 und 6 zur Berechnung des Schlüssels verwendet.

Mittels dynamischer Analyse besteht die Möglichkeit der Beobachtung der Methode zur Laufzeit, um gültige Schlüsselparameter zu ermitteln und um eine eigene Implementierung des Pseudocodes zu erstellen bzw. zu testen.

Der von Miller und Valasek ermittelte Algorithmus zur Authentifizierung gegenüber Ford-Fahrzeugen [miva] konnte ebenfalls ermittelt und das Ergebnis somit bestätigt werden. Im Anhang dieser Arbeit befindet sich der Algorithmus in Form von IDA-Pseudocode (siehe 8). Dieser Algorithmus wurde von Miller und Valasek erfolgreich mit einem Ford Explorer aus dem Jahre 2010 getestet.

Darüber hinaus konnten zwei weitere potentielle Schlüsselalgorithmen identifiziert werden, deren Anwendungsgebiet momentan jedoch unbekannt ist. Einer der Algorithmen ist im folgenden Listing zu sehen:

```
1 return ((unsigned __int8)key ^ (key >> 8) ^ 0x9B) + 0xA932;
```

Listing 18: Analyseergebnisse: Unbekannter potentieller Authentifizierungsalgorithmus (1)

Es ist zu sehen, dass die Komplexität der verwendeten Algorithmen stark schwankt. Oft werden zur Berechnung des Schlüssels keinerlei Seed-Werte einbezogen und statische Werte zur Berechnung verwendet. Der zweite extrahierte Algorithmus kann im Anhang eingesehen werden [8](#).

### 5.2.2 XX-Software

Es wurde eine weitere ECU-Programmieranwendung analysiert, welche von einem Drittanbieter mitsamt kompatibler Hardware vertrieben wird. Diese Software dient als Multi-funktionswerkzeug für Fahrzeuge verschiedener Automobilhersteller mit zahlreichen Steuergerättypen.

Mittels der Analyse durch *IDA* konnten die *potentiellen* Authentifizierungsalgorithmen der Steuergeräte XXX und XXX aufgefunden werden, welche von der Firma XXX hergestellt werden.

Bereits Alberto Garcia Illera und Javier Vazquez Vidal ermittelten diese Algorithmen, welche sie auf der Defcon-21-Konferenz zum Großteil veröffentlichten [\[ajr\]](#). Die in der XXX-Software enthaltenen Algorithmen befinden sich im Anhang dieser Arbeit in Form von Pseudocode ([8 ff.](#)). Enthalten ist ein weiterer ECU-Schlüssel für das XXX-Steuergerät welcher bei der Konferenz-Veröffentlichung nicht publiziert wurde. Auffällig ist bei den verwendeten Algorithmen die Verwendung von statischen internen Schlüsseln, wie zum Beispiel 12233 (0x2FC), und die Verwendung von Bitshift-Befehlen.

Diese Reverse-Engineering-Maßnahmen machen deutlich, dass vermeintlich geheime Schlüssel und Algorithmen durchaus aus Software extrahiert werden können. Neben der Analyse der Steuergerät-Firmware bietet diese Methode einen Einblick in die verwendeten Authentifizierungsprotokolle. Durch die Offenlegung der Verfahren wird unter Umständen ein vollständiger Zugriff auf interne Systeme von Fahrzeugen möglich. Dies erlaubt unerwünschte Manipulationen von Automobilien.

## 5.3 Weitere Ergebnisse

Aus Gründen der Übersicht und Vollständigkeit werden innerhalb dieses Abschnitts einige Ergebnisse bereits veröffentlichter Analysen aufgegriffen und zusammenfassend erläutert. Der Fokus liegt auf den Ergebnissen von Miller und Valasek [\[miva\]](#), welche durch Analysen eines Toyota Prius erhoben wurden. Diese Beispiele von Befunden demonstrieren, dass mit ausreichend Zeit und Ressourcen durchaus kritische Ergebnisse aus Fahrzeugsystemen erhoben werden können. Zur Analyse wurden zahlreiche bereits innerhalb dieser Arbeit beschriebene Methoden verwendet [\[mvd\]](#). Die folgenden Aspekte stellen einige der ersten [\[ewc\]](#) für die Öffentlichkeit einsehbaren [\[agw\]](#) Sicherheitslücken von modernen Fahrzeugen dar.

### 5.3.1 Einsatz von Prüfsummen

Es besteht die Möglichkeit, dass innerhalb eines Fahrzeugnetzwerks proprietär berechnete Prüfsummen für jedes CAN-Paket verwendet werden. Dies hat zur Folge, dass Pakete mit inkorrekt er Prüfsumme von Steuergeräten verworfen und ignoriert werden. Weitergehende Analysen sind somit nur mit der Kenntnis der herstellerspezifischen Prüfsumme möglich. Im Falle des Toyota Prius berechnet sich die eingesetzte Prüfsumme wie folgt:

$$ID + Länge + (Daten[0] - Daten[Länge - 2]) \& 0xFF$$

Das Ergebnis der Prüfsummenberechnung stellt im analysierten Fahrzeug stets das letzte Byte der Nutzdaten dar.

### 5.3.2 Einleiten vom Bremsvorgängen

Miller und Valasek fanden im selben Fahrzeug diverse Möglichkeiten in für den Fahrer kritische Systeme einzugreifen. Eine davon umfasst die Betätigung der Bremse. In modernen Fahrzeugen sind oft die Beschleunigung, Bremse und Lenkung nicht nur manuell durch den Fahrer steuerbar, sondern auch durch Assistenzsysteme. Im Falle des analysierten Fahrzeugs wurde das *Pre-Collision-Assistenzsystem* betrachtet. Dieses leitet automatisch einen Bremsvorgang ein, sobald über ein Radarsystem ein Fahrzeug in Richtung der Fahrzeugfront erkannt wird. Dies beschreibt ein oft anzutreffendes Szenario: Vernetzte Sensorsysteme erkennen einen Handlungsbedarf seitens der Kernsysteme des Fahrzeugs. Daraufhin werden Busnachrichten versandt, um diese Kernsysteme anzuweisen bestimmte Handlungen durchzuführen. So benachrichtigt im Falle des Toyota Prius das Radarsystem das Bremssystem, um einen Bremsvorgang einzuleiten.

Diese Benachrichtigung geschieht in Form eines einzelnen CAN-Pakets:

*ID : 283, Länge : 07, Daten : 61 00 E0 BE 8C 00 17*

Der Wert des ersten Bytes des Datensegments muss bei jedem Versenden inkrementiert und mit 0x7F Und-Verknüpft werden, da dieses Byte eine Sequenznummer enthält.

Sofern dieses Paket wiederholt auf dem CAN-Bus versandt wird, kann das Fahrzeug *bei voller Fahrt* gestoppt werden. Dabei ist die momentane Geschwindigkeit nicht von Bedeutung. Zudem kann das Fahrzeug an der Weiterfahrt gehindert werden: Bei der andauernden Versendung dieses Pakets auf dem Bus kann selbst bei vollständig betätigtem Gaspedal keine Beschleunigung durchgeführt werden.

### 5.3.3 Aufrechterhaltung der Beschleunigung

Das Gaspedal des Toyota Prius ist nicht direkt mit dem Motorsteuergerät verbunden. Stattdessen ist es mit einem Verwaltungssteuergerät verbunden. Dies erfolgt aufgrund des Hybridantriebs des Fahrzeugs: Auf diese Art und Weise können verschiedene Motoren

angesteuert werden. Das Verwaltungssteuergerät erzeugt bei Betätigung des Pedals CAN-Pakete, welche vom Motorsteuergerät verarbeitet werden – das Fahrzeug beschleunigt. Folgendes CAN-Paket weißt das Motorsteuergerät an, eine Beschleunigung durchzuführen:

*ID : 037, Länge : 07, Daten : C7 17 58 13 9D 00 24*

Durch das Einschleusen dieses CAN-Pakets kann eine zuvor gestartete Beschleunigung über einige Sekunden ohne Betätigung des Gaspedals aufrecht erhalten werden. Eine komplettete Steuerung der Beschleunigung ist aufgrund des komplexen Hybridantriebs mit diesem Paket alleine nicht möglich.

#### 5.3.4 Eingriff in elektronische Lenkungen

Automatische Einparkssysteme erfordern eine elektronisch ansteuerbar Lenkung. Diese wird im Falle des Toyota Prious durch einen Servomechanismus beim automatischen Einparken gesteuert. Mithilfe von drei verschiedenen Paketen kann die Lenkung durch einen Angreifer *bei voller Fahrt* kontrolliert werden:

- Ansteuerungspaket für den Servomechanismus. Hier kann der gewünschte Einstellungswinkel des Lenkrads eingestellt werden.
- Rückwärtsgang-Broadcastpaket: Der Servomechanismus wird aus Sicherheitsgründen nur betätigt, falls der Rückwärtsgang eingelegt wurde. Um jedoch auch in allen anderen Gängen das Lenkrad kontrollieren zu können, muss ständig ein bestimmtes Paket in den Bus eingeschleust werden. Dieses Paket sagt aus, dass der Rückwärtsgang eingelegt ist – auch wenn dies momentan nicht der Fall ist.
- Geschwindigkeitspaket: Ein weiterer Sicherheitsmechanismus verlangt eine Geschwindigkeit von unter 4 Meilen pro Stunde, um das Lenkrad elektronisch anzusteuern. Über ein ständig in den Bus versandtes Geschwindigkeitspaket kann dem Fahrzeug vorgetäuscht werden, dass die Geschwindigkeit im vorgeschriebenen Bereich liegt.

Durch die Kombination aller drei Pakete kann das Lenkrad bei jeder Geschwindigkeit und in jedem Gang kontrolliert werden.

#### 5.3.5 Statischer UDS-Seed

Um für eine Diagnosesitzung höhere Privilegien anzufordern, wird für das UDS-Protokoll ein Challenge-Response-Verfahren durchgeführt (vgl. 2.4). Die Generierung des Seeds wird über einen proprietären Algorithmus abgewickelt, welcher essenziell für die Sicherheit des Fahrzeugs ist. Umso kritischer ist der Umstand, dass bei Ford Escapes aus dem Jahr 2010 bei jedem Authentifizierungsversuch der selbe Seed, nämlich 11 22 33, vom Fahrzeug generiert wird. Hierdurch wird unter anderem eine Brute-Force-Attacke des daraus resultierenden 24-Bit-Werts ermöglicht. Detailliertere Analysen der Fahrzeug-Firmware zeigen,

dass der Programmcode zur zufälligen Generierung von Seed-Werten existiert und zudem ausgeführt wird. Jedoch wird vor dem `return`-Statement der generierte Wert mit dem statischen Standard-Seed überschrieben – höchstwahrscheinlich eine Debugging-Maßnahme. Es wurde möglicherweise versäumt diesen Entwicklungscode aus der produktiv eingesetzten Version zu entfernen.

## CAN-Matrix des analysierten Kombiinstruments

Tabelle 8: Reverse-Engineering eines Kombiinstruments: Ermittelte CAN-Matrix (1)

ID	Nutzdaten	Bezeichnung
040	0000000001000000	Gurtwarnung aktiviert
040	0000000000000000	Gurtwarnung deaktiviert
101 & 308	0000000000000000 bzw. 0000X <sub>1</sub> X <sub>2</sub> X <sub>3</sub> X <sub>4</sub> 00000000	Einstellung der Geschwindigkeit: Beide Pakete müssen kombiniert werden Eine Erhöhung von $X_i$ um den Wert 1 bewirkt eine Veränderung von $x$ km/h: $X_1 \approx 0,5$ km/h $X_2 \approx 0,01$ km/h $X_3 \approx 67$ km/h $X_4 \approx 4,1$ km/h
30D	0001000000000000	Parkbremse (grün)
30D	0400000000000000	Parkbremse (rot)
363	0000440000000000	Blinker links aktiviert
363	0000F80000000000	Blinker rechts aktiviert
363	FFFFFFFFFFFFFFFFFF	Blinker links und rechts aktiviert
363	0000000000000000	Alle Blinker deaktiviert
397	0000000000000020	Spurassistent (gelb)
397	0000000000000050	Spurassistent (grün)
3C0	00000200	Zündung aktiviert
3C0	00000100	Zündung deaktiviert
3C0	BC204007A5BCB8	Aufblinken aller Symbole
585	00020000000000	Anzeige von $TR$
590	00000000000D0000	Tageskilometerzähler zeigt REDACTED
590	0000000000020000	Anzeige von $L1$ unter dem Tageskilometerzähler (Vermutung: Nummer des neu angelernten Schlüssels)
590	00000000000F0000	Anzeige von $L1\ 2/2$ unter dem Tageskilometerzähler
590	00000000000F0000	Anzeige von $L3$ unter dem Tageskilometerzähler
5F0	222222222222	Dimmen des Displays Nutzdaten entsprechen dem Helligkeitswert

Tabelle 9: Reverse-Engineering eines Kombiinstruments: Ermittelte CAN-Matrix (2)

ID	Nutzdaten	Bezeichnung
5F0 & 662	FFFFFFFFFFFFFFF bzw. 00000F0000000000	Fernlicht an
5F0 & 662	FFFFFFFFFFFFFFF bzw. 00000000B0000000	Fernlichtautomatik an
661	0002000000000000	Anzeige von drei grünen Pfeilen
663	0400000X <sub>1</sub> 00000000	Anzeige von <i>TR</i> mit Prozentzahl bzw. Strichen: $X_1 \in \{0 \dots D\}$ : Anzeige einer Prozentzahl $X_1 \in \{E, F\}$ : Anzeige von — Paket muss 2x versandt werden, sonst nur Anzeige von <i>TR</i> allein
700 & 714	0210030000000000	Start Programming-Session
700 & 714	0210400000000000	Start Extended-Session
714	0210X <sub>1</sub> X <sub>2</sub> 0000000000	Starten von proprietären Sitzungen
714	0211820000000000	Zurücksetzen des Geräts
714	0722225F84E15D03	Auslesen eines Speicherfragments

## Skript zur Geschwindigkeitskonvertierung

Dieses Skript kann im Kombination mit den bereits ermittelten CAN-Paketen verwendet werden, um die Geschwindigkeit des Tachometers auf den gewünschten Wert einzustellen. Hierzu konvertiert dieses Skript die eingegebene Geschwindigkeit in CAN-Nutzdaten.

```
1 #! /usr/bin/python3
2
3 from time import sleep
4 from math import ceil
5 from random import shuffle
6 import codecs
7
8 # This script converts the desired displayed speed for an
9 # <redacted :D> speedometer to a CAN payload
10
11 # @author pschmied
12
13 #####
14 # SETTINGS #
15 # calculation value and index in packet
16 firstFineTuningCalc = (0.5, 4)
17 secondFineTuningCalc = (0.01, 5)
18 firstByteCalc = (67, 6)
19 secondByteCalc = (4.1, 7)
20 calcVals = [firstByteCalc,
21             secondByteCalc,
22             firstFineTuningCalc,
23             secondFineTuningCalc]
24 #####
25
26 def setChar(string, char, idx):
27
28     """
29         Set the character at index ``idx`` of the string ``string``
30         to the character ``char``.
31     """
32
33     stringList = list(string)
34     stringList[idx] = char.upper()
35     return "".join(stringList)
36
37 def kmhToData(kmh):
38
39     """
40         This converts the desired displayed speed, which is stored in
41         ``kmh`` as integer, to a CAN payload.
42     """
43
44     data = "0" * 16
```

```

45
46     for calcVal in calcVals:
47         while kmh >= calcVal[0]:
48             currentVal = int(data[calcVal[1]], 16)
49             if currentVal == 15:
50                 newVal = "F"
51                 # check if its possible to increment the previous value
52                 if calcVal == secondByteCalc:
53                     # set current value to 0 and increment previous value
54                     newVal = "0"
55                     currentValPrevious = int(data[firstByteCalc[1]], 16)
56                     newValPrevious = hex(currentValPrevious + 1) \
57                         .split("0x")[1]
58                     data = setChar(data, newValPrevious, firstByteCalc[1])
59             else:
60                 newVal = hex(currentVal + 1).split("0x")[1]
61                 data = setChar(data, newVal, calcVal[1])
62                 kmh -= calcVal[0]
63
64     print(data)
65
66 kmh = 220
67 kmhToData(kmh)

```

Listing 19: Analyseergebnisse: Skript zur Geschwindigkeitskonvertierung

## Diagnosedienste des analysierten Kombiinstruments

Tabelle 10: Reverse-Engineering eines Kombiinstruments: Diagnosedienste

CAN-ID	UDS-SID	Dienstbezeichnung	Funktion (ISO 14230)
700	0x10	Diagnostic Session Control	Starten einer Diagnosesitzung
700	0x11	ECU Reset	Zurücksetzen des Steuergeräts
700	0x14	Clear Diagnostic Information	Leerung des Fehlerspeichers
700	0x19	Read DTC Information	Auslesen des Fehlerspeichers
700	0x22	Read Data By Identifier	Auslesen von Steuergerätdaten via ID
700	0x2E	Write Data By Identifier	Schreiben von Steuergerätdaten via ID
700	0x2F	Input/Output Control By Identifier	Feste Werte von Ein- und Ausgängen setzen
700	0x3E	Tester Present	Keepalive-Service
714	0x10	Diagnostic Session Control	Startet Diagnosesitzung
714	0x11	ECU Reset	Zurücksetzen des Steuergeräts
714	0x14	Clear Diagnostic Information	Leerung des Fehlerspeichers
714	0x19	Read DTC Information	Auslesen des Fehlerspeichers
714	0x22	Read Data By Identifier	Auslesen von Steuergerätdaten via ID
714	0x23	Read Memory By Address	Auslesen des Speichers via Speicheradresse
714	0x27	Security Access	Schlüsselaustausch (Challenge-Response)
714	0x28	Communication Control	Verbietet nicht-diagnosebezogene CAN-Nachrichten
714	0x2E	Write Data By Identifier	Schreiben von Steuergerätdaten via ID
714	0x2F	Input/Output Control By Identifier	Feste Werte von Ein- und Ausgängen setzen
714	0x31	Routine Control	Starten von im Steuergerät definierten Routinen
714	0x34	Request Download	Download der Firmware (Von Diagnosegerät zu ECU)
714	0x36	Transfer Data	Übertragung von Daten
714	0x37	Request Transfer Exit	Fertigstellung der Übertragung
714	0x3D	Write Memory By Address	Schreiben des Speichers via Speicheradresse
714	0x3E	Tester Present	Keepalive-Service
714	0x85	Control DTC Settings	Verhindert Anlegung von Fehlereinträgen im Speicher

## Vermutete XXX ECU-Schlüssel

Tabelle 11: Vermutete ECU-Schlüssel (Teil 1 – ASCII-Schlüssel)

<b>Hexadezimalwerte</b>	<b>ASCII-Wert</b>	<b>Hexadezimalwerte</b>	<b>ASCII-Wert</b>
41 49 53 49 4E	AISIN	41 52 49 41 4E	ARIAN
42 4F 53 45 58	BOSEX	42 6F 73 63 68	Bosch
42 72 61 64 57	BradW	43 4F 4C 49 4E	COLIN
44 49 4F 44 45	DIODE	44 52 49 46 54	DRIFT
45 4D 45 31 53	EME1S	46 6C 61 73 68	Flash
48 41 5A 45 4C	HAZEL	4A 41 4D 45 53	JAMES
4A 61 6E 69 73	Janis	4A 65 73 75 73	Jesus
4C 75 70 69 6E	Lupin	4D 41 5A 44 41	MAZDA
4D 61 7A 64 41	MazdA	4F 53 4E 45 44	OSNED
50 41 4E 44 41	PANDA	50 61 44 6D 41	PaDmA
52 45 4D 41 54	REMAT	52 6F 77 61 6E	Rowan
53 48 4F 57 41	SHOWA	54 41 4D 45 52	TAMER
54 4F 42 42 45	TOBBE	54 52 4F 48 53	TROHS
61 5F 62 61 64	a_bad	6D 41 5A 44 61	mAZDa
6E 6F 77 61 52	nowaR	62 68 61 77 6E	bhawn
63 6F 6E 74 69	conti		

Tabelle 12: Vermutete ECU-Schlüssel (Teil 2)

Hexadezimalwerte			
0A 6D C1 3B 20	0D 5A DB DF CA	0E AA 0B A0 16	11 22 33 44 55
11 4D 87 75 AF	12 13 19 76 30	12 23 34 45 56	13 4B 7C F3 5C
14 20 E6 89 0A	15 1F 52 1F 7F	15 24 33 42 51	16 27 38 49 5A
16 65 2C EA BE	16 B9 13 77 70	17 46 CE B4 35	17 7B 6A 96 74
18 AE 78 4A D2	19 AC 3A 1C C2	19 D3 0B 2F D2	1A 12 D3 98 49
1A 2B 3C 4D 5E	1B 44 14 4D 24	1C 27 50 76 77	1D 61 E5 C7 0D
1F 1A 4F 3E E6	20 4A FE 9C 2D	20 AC 71 5F F7	20 E4 48 E1 D1
22 5F B4 AC 54	22 70 EA 4C 11	22 7B 3F 23 77	22 A1 60 51 A9
23 11 D2 A2 67	24 31 DE F9 46	24 87 64 65 85	24 87 64 70 32
25 92 CA 34 29	26 81 45 29 7D	26 8C B4 71 EE	27 76 76 59 C6
27 A1 2B 19 83	28 81 45 29 7D	28 E2 22 63 5F	29 64 B2 72 C2
2C D8 73 A9 14	2D 86 C5 57 A1	2E 67 CB 8A 30	31 32 33 34 35
31 49 21 63 27	31 63 46 66 48	34 1F 3C FB C5	34 23 39 11 77
34 D9 52 C9 42	35 E7 FA 6D D1	37 E1 A2 15 C8	38 89 85 87 3A
3A 62 93 D6 F7	3A 8E CF 9C CE	3C E2 DB 41 9A	3D 80 49 83 B3
3F 37 5F 22 59	3F 43 EF 74 BE	41 51 26 84 11	41 57 54 43 55
41 82 F6 78 EE	41 AA 42 BB 43	42 43 4D 59 32	42 A4 83 99 79
42 D3 6E 34 CE	42 DD 6E 34 CE	44 DD 45 EE 46	45 55 43 44 31
46 4D 43 30 31	46 6E 74 63 4D	47 76 66 65 52	47 A7 3B 83 62
48 12 A6 D5 7B	48 53 54 43 4D	48 87 29 88 9D	49 76 66 65 52
49 BE 98 2A 14	4A 36 31 43 46	4B 30 32 31 36	4C 43 41 42 4C
4D 48 65 71 79	4E 53 59 4E 53	4E 8A E2 CA D8	50 C8 6A 49 F1
50 C8 6A 49 F2	51 1B 53 74 74	51 9A 72 13 1C	53 4B 65 21 AD
53 55 4D 2D 33	53 67 98 B3 A4	54 75 BC 4E 68	57 A2 F7 C3 49
58 35 82 C8 1D	59 5B 58 72 52	5A 3B 51 4A 35	5A 89 E4 41 72
5B 41 74 65 7D	5C 55 28 9B 6D	5C EB 11 7B 30	5E 10 0F 46 33
5E 18 BF 9B 75	5F 7D F5 F7 93	5F 9C 99 A3 50	61 42 2B 54 64
61 EB AD C6 24	62 48 C5 A2 5F	64 56 36 4C A7	65 38 74 36 31
66 6F 61 77 65	69 69 B6 53 A1	6A 55 68 51 5E	6A A5 68 56 5E
6B A7 24 0A 71	6B A7 43 0A 71	6C 2E A0 71 E6	6C 5A B3 C7 8B
6F 6E 69 62 68	71 18 41 69 68	71 18 A5 68 68	71 48 77 B2 4B
71 55 21 41 C2	71 55 21 5E A5	71 55 21 A5 5E	75 37 BB D8 89
75 B1 56 83 4A	76 66 84 57 8C	77 87 A5 86 A3	78 77 68 6B 53
79 69 96 56 B6	79 B6 8C A4 64	7B 73 77 6A A5	7B 87 89 9B 57
7C 70 D0 30 C1	7D D0 C4 76 62	83 6A CF 36 1B	83 9E E2 12 E0
85 77 C8 65 36	85 EF F9 F5 DC	86 9A 78 4D 90	87 C8 8B 77 A6
88 2A 7B 93 36	88 2A 7B 93 54	88 6D 91 75 7D	88 99 96 6A 5C
89 D5 7F B3 A7	8A 78 90 34 F7	8A 81 A8 88 82	8C 4A D2 1F 2E
8C 54 F8 0B D7	8C F5 A5 19 EE	92 77 6B 88 77	93 46 98 48 B9

Tabelle 13: Vermutete ECU-Schlüssel (Teil 3)

Hexadezimalwerte			
94 87 A9 A6 7A	96 87 88 94 A8	96 99 56 94 85	96 A2 3B 83 9B
97 62 79 84 EC	97 7A D9 3C 82	98 85 98 77 A9	98 97 68 77 AA
99 77 88 88 77	9A 78 56 34 12	9A B6 99 6C 9A	9C CA 8A 7A 37
9D 6C C3 12 BB	9F 75 4B 3B 87	A2 A3 AC CA 50	A3 B2 C0 14 92
A3 C1 E4 19 89	A5 B3 EF DA 76	A5 FD 4C 45 DA	A6 37 9E 84 34
A7 6C B2 79 AA	A7 C2 E9 19 92	A7 C2 E9 2C 7A	A8 6B 9C 87 68
AA 5A 35 BE D4	AA 76 88 6B A7	AA 77 5C 45 B7	AA 7C 3A BD D9
AA BB CC DD EE	AA CC CC 33 55	AB 7D D1 D2 71	AB DC 74 45 C6
AC F7 29 92 E1	AD D9 A2 67 75	AE C7 B4 4B AC	B2 6D 74 9A 57
B3 97 C8 6A 37	B5 0A B4 96 2C	B6 45 17 77 D3	B6 E3 D7 7C 3D
B8 3F 24 7F FF	C1 B0 CA D0 1A	C5 A4 61 A4 A7	C8 9B 10 48 8D
C8 B2 E3 9E 76	C9 EA 84 CB 10	CA 5B 1B 48 AF	CB 41 12 28 71
D1 F3 2D 91 4B	D8 41 5D 77 A5	DA CC CB 6E B3	DC 0D E5 B1 AB
DC 40 0A 59 23	DC D5 44 7A E6	DD 16 1A 48 AF	DF 3A 14 69 C2
E1 84 BE 23 29	E6 76 3F ED 74	E7 29 D1 4B 41	EF E5 7A 91 6A
F3 11 45 4C 73	F4 31 0F B2 52	F4 79 1A 60 CB	F5 1E 14 87 C0
F7 32 D7 3A 12	F8 39 12 96 0E	FA6B 5A 70 47	FD 4C 92 81 ED
FD B3 F4 35 88	FE 42 28 D3 AD	FF FF FF FF FF	

## ECU-Authentifizierungsalgorithmen

**REDACTED**

```

1 v9 = 0xC541A9;
2 for ( i = 0; i < 32; ++i ) {
3   v6 = (((((key & 0xFF0000) >> 16) | key & 0xFF00
4   | (seed_1 << 24) | ((unsigned __int8)key << 16)) >> i)
5   & 1 ^ v9 & 1) << 23;
6
7   v12 = (v6 | (v9 >> 1)) & 0xEF6FD7 | (((((v6 | (v9 >> 1))
8   & 0x100000) >> 20) ^ (((v6 | (v9 >> 1)) & 0x800000) >> 23))
9   << 20) | (((((v9 >> 1) & 0x8000) >> 15) ^ (((v6 | (v9 >> 1))
10  & 0x800000) >> 23)) << 15) | (((((v9 >> 1) & 0x1000) >> 12)
11  ^ (((v6 | (v9 >> 1)) & 0x800000) >> 23)) << 12) |
12  32 * (((((v9 >> 1) & 0x20) >> 5) ^ (((v6 | (v9 >> 1))
13  & 0x800000) >> 23)) | 8 * (((v9 >> 1) & 8) >> 3)
14  ^ (((v6 | (v9 >> 1)) & 0x800000) >> 23));
15
16
17   v9 = (v6 | (v9 >> 1)) & 0xEF6FD7 | (((((v6 | (v9 >> 1))
18  & 0x100000) >> 20) ^ (((v6 | (v9 >> 1)) & 0x800000) >> 23))
19  << 20) | (((((v9 >> 1) & 0x8000) >> 15) ^
20  (((v6 | (v9 >> 1)) & 0x800000) >> 23)) << 15) |
21  (((((v9 >> 1) & 0x1000) >> 12) ^ (((v6 | (v9 >> 1)))

```

```

22     & 0x800000) >> 23)) << 12) | 32 * (((v9 >> 1) & 0x20) >> 5) ^
23     (((v6 | (v9 >> 1)) & 0x800000) >> 23)) | 8 *
24     (((((v9 >> 1) & 8) >> 3) ^ (((v6 | (v9 >> 1)) & 0x800000)
25     >> 23));
26 }
27
28 for ( j = 0; j < 32; ++j )
29 {
30     v7 = (((((seed_5 << 24) | (seed_4 << 16) | seed_2 |
31     (seed_3 << 8)) >> j) & 1 ^ v9 & 1) << 23;
32     v12 = (v7 | (v9 >> 1)) & 0xEF6FD7 | (((((v7 |
33     (v9 >> 1)) & 0x100000) >> 20) ^ (((v7 | (v9 >> 1))
34     & 0x800000) >> 23)) << 20) | (((((v9 >> 1) & 0x8000) >> 15)
35     ^ (((v7 | (v9 >> 1)) & 0x800000) >> 23)) << 15)
36     | (((((v9 >> 1) & 0x1000) >> 12) ^ (((v7 | (v9 >> 1))
37     & 0x800000) >> 23)) << 12) | 32 * (((v9 >> 1) & 0x20) >> 5)
38     ^ (((v7 | (v9 >> 1)) & 0x800000) >> 23)) | 8
39     * (((v9 >> 1) & 8) >> 3) ^ (((v7 | (v9 >> 1)) & 0x800000)
40     >> 23));
41
42     v9 = (v7 | (v9 >> 1)) & 0xEF6FD7 | (((((v7 | (v9 >> 1))
43     & 0x100000) >> 20) ^ (((v7 | (v9 >> 1)) & 0x800000) >> 23))
44     << 20) | (((((v9 >> 1) & 0x8000) >> 15) ^ (((v7 | (v9 >> 1))
45     & 0x800000) >> 23)) << 15) | (((((v9 >> 1) & 0x1000) >> 12)
46     ^ (((v7 | (v9 >> 1)) & 0x800000) >> 23)) << 12) | 32 *
47     (((v9 >> 1) & 0x20) >> 5) ^ (((v7 | (v9 >> 1))
48     & 0x800000) >> 23)) | 8 * (((v9 >> 1) & 8) >> 3)
49     ^ (((v7 | (v9 >> 1)) & 0x800000) >> 23));
50 }
51
52 return ((v12 & 0xF0000) >> 16) | 16 * (v12 & 0xF)
53 | (((((v12 & 0xF0000) >> 20) | ((v12 & 0xF000) >> 8)) << 8)
54 | ((v12 & 0xFF0) >> 4 << 16);

```

Listing 20: Analyseergebnisse: Authentifizierungsalgorithmus für Ford-Fahrzeuge

## Unbekannter Authentifizierungsalgorithmus

Dieser Algorithmus wird möglicherweise bei einigen Fahrzeugen der AA, BB, CC oder DD verwendet:

```
1 if ( BYTE1(key) & 1 )
2 {
3     v4 = 4;
4     v3 = 6;
5 }
6 else
7 {
8     v4 = 0;
9     v3 = 2;
10}
11
12 v1 = changeKey((key >> 8) & 0xFF, (unsigned __int8)key,
13   ((signed int)(unsigned __int8)key >> v4) & 3);
14
15 return v1 | (unsigned __int16)((unsigned __int8)changeKey(
16   (key >> 16) & 0xFF,
17   (unsigned __int8)key,
18   ((signed int)(unsigned __int8)key >> v3) & 3) << 8);
```

Listing 21: Analyseergebnisse: Unbekannter Authentifizierungsalgorithmus (2)

Die Methode `changeKey` verändert den eingesetzten Schlüssel anhand statischer Merkmale und führt Additionen und Multiplikationen durch. Folgendes Beispiel stellt ein Beispiel einer solchen Operation dar:

```
1 v4 = a1 + 13 * (3 * a1 + 2 * a2) * 13 * (3 * a1 + 2 * a2) [...]
```

Listing 22: Analyseergebnisse: Unbekannter Authentifizierungsalgorithmus (2): changeKey

## REDACTED

```
1 BYTE1(v5) = *(_BYTE *)seed_4;
2 HIWORD(v6) = 0;
3 BYTE1(v6) = *(_BYTE *) (seed_4 + 2);
4 LOBYTE(v5) = *(_BYTE *) (seed_4 + 1);
5 LOBYTE(v6) = *(_BYTE *) (seed_4 + 3);
6 v7 = *(_WORD *) (key + 0x2508E0);
7 LOWORD(seed_1) = *(_WORD *) (seed_1 + 0x2508E0) >> 1;
8 v8 = 2 * seed_1;
9 if ( v7 == (_WORD)v8 )
10 {
11     internal_key1 = 0xF25E; // Vermutung: Schlüssel für
12     internal_key2 = 0x6533; // verschiedene Zugriffsmodi
13 }
14 else
15 {
16     internal_key1 = 0x4D41;
17     internal_key2 = 0x5443;
18 }
19 v10 = 5;
20 do
21 {
22     v11 = v5 & 0x8000;
23     v12 = 2 * v5;
24     if ( v11 )
25     {
26         v13 = 2 * v6;
27         LOWORD(v8) = (unsigned __int8)(2 * v6) | 1;
28         LOWORD(v13) = (unsigned __int16)v6 >> 15;
29         v8 = internal_key1 ^ (2 * v6 | v8);
30         v6 = v8;
31         v5 = internal_key2 ^ (v12 & 0xFFFF | v13);
32     }
33     else
34     {
35         LOWORD(v5) = ((unsigned __int16)v6 >> 15) | v5 & 0xFFFF;
36         v6 *= 2;
37     }
38     --v10;
39 }
40 while ( v10 ); // for-Schleife mit 5 Durchläufen
41 result = BYTE1(v5);
42 return result;
```

Listing 23: Analyseergebnisse: Potentieller EDC15-Authentifizierungsalgorithmus

## **REDACTED**

```
1 result = *(_BYTE *) (key + 0x25005C)
2     + ((*(_BYTE *) (key + 0x25005B) +
3         ((*(_BYTE *) (key + 0x25005A) +
4             (*(_BYTE *) (key + 0x250059) << 8)) << 8))
5     << 8) + 0x2FC9;
```

Vorhergehenden Arbeiten kann entnommen werden, dass die Addition der Hexadezimalwerte mit dem Muster `0x0x25005*` eventuell nicht benötigt wird [ajr].

Der durch die Authentifizierung gewährte Zugriff bietet einen lesenden Zugriff auf das Steuergerät. Für einen schreibenden Zugriff wird ein Level-1-Zugriff benötigt [jve]. Der hierfür zu verwendende Algorithmus kann [ajr] entnommen werden.

# Literatur

- [agw] Andy Greenberg (Wired). *Hackers Remotely Kill a Jeep on the Highway—With Me in It.* Eingesehen am 09.08.2017 08:51 Uhr. URL: <https://www.wired.com/2015/07/hackers-remotely-kill-jeep-highway/>.
- [ajr] Javier Vazquez Vidal Alberto Garcia Illera. *Dude, WTF in my car? (Defcon 21)*. Eingesehen am 19.04.2017 14:41 Uhr. URL: <https://www.youtube.com/watch?v=K2lJr5ccZBQ>.
- [asi] Alexey Sintsov. *CANToolz - framework for black-box CAN network analysis.* Eingesehen am 03.04.2017 09:35 Uhr. URL: <https://github.com/eik00d/CANToolz>.
- [asvg] Openclipart. *Weißes Auto.* Eingesehen am 20.03.2017 15:32 Uhr. URL: <http://publicdomainvectors.org/de/kostenlose-vektorgrafiken/Wei%C3%9Fes-Auto/49183.html>.
- [bgb] Ben Gale (Broadcom Corporation). *Ethernet security in the car.* Eingesehen am 25.07.2017 11:00 Uhr. URL: [https://standards.ieee.org/events/automotive/2014/19\\_Ethernet\\_Car\\_Security.pdf](https://standards.ieee.org/events/automotive/2014/19_Ethernet_Car_Security.pdf).
- [brp] Portswigger. *Burp Suite.* Eingesehen am 13.07.2017 13:56 Uhr. URL: <https://portswigger.net/burp/>.
- [cbu] GitHub-User CaringCaribou. *caringcaribou: A friendly car security exploration tool.* Eingesehen am 25.04.2017 14:29 Uhr. URL: <https://github.com/CaringCaribou/caringcaribou>.
- [cfr] Wikipedia-User Endres commonswiki. *CAN-Frame mit Pegeln mit Stuffbits.* Eingesehen am 24.03.2017 10:07 Uhr. URL: [https://de.wikipedia.org/wiki/Datei:CAN-Frame\\_mit\\_Pegeln\\_mit\\_Stuffbits.svg](https://de.wikipedia.org/wiki/Datei:CAN-Frame_mit_Pegeln_mit_Stuffbits.svg).
- [cia] Friedhelm Greis (Golem.de). *Vault 7 – Was macht die CIA mit gehackten Autos?* Eingesehen am 10.03.2017 11:02 Uhr. URL: <https://www.golem.de/news/vault-7-was-macht-die-cia-mit-gehackten-autos-1703-126639.html>.
- [cic] Cisco. *Network Topology Icons - Doing Business With Cisco.* Eingesehen am 10.03.2017 12:51 Uhr. URL: <http://www.cisco.com/c/en/us/about/brand-center/network-topology-icons.html>.
- [crb] Robert Bosch GmbH. *CAN Specification (Version 2).* Eingesehen am 23.03.2017 13:02 Uhr. URL: [http://www.bosch-semiconductors.de/media/ubk\\_semiconductors/pdf\\_1/canliteratur/can2spec.pdf](http://www.bosch-semiconductors.de/media/ubk_semiconductors/pdf_1/canliteratur/can2spec.pdf).
- [csa] Craig Smith. *ICSim (Instrument Cluster Simulator).* Eingesehen am 05.04.2017 14:36 Uhr. URL: <https://github.com/zombieCraig/ICSim>.
- [csb] Craig Smith. *Exiting the Matrix: Introducing Metasploit's Hardware Bridge.* Eingesehen am 05.04.2017 09:30 Uhr. URL: <https://community.rapid7.com/community/transpo-security/blog/2017/02/02/exiting-the-matrix>.

- [csc] Craig Smith. *UDSim (UDS Simulator and Fuzzer)*. Eingesehen am 05.04.2017 16:18 Uhr. URL: <https://github.com/zombieCraig/UDSim>.
- [cut] Marc Kleine-Budde Oliver Hartkopp. *can-utils: Linux-CAN / SocketCAN user space applications*. Eingesehen am 04.04.2017 14:05 Uhr. URL: <https://github.com/linux-can/can-utils>.
- [dev] 8devices. *USB2CAN*. Eingesehen am 03.04.2017 16:02 Uhr. URL: <http://www.8devices.com/products/usb2can/>.
- [devg] GitHub-User krumboeck. *CAN driver for UAB "8 devices USB2CAN converter*. Eingesehen am 02.05.2017 08:29 Uhr. URL: <https://github.com/krumboeck/usb2can>.
- [diag] OBD Diagnose Shop. *Maxidas DS708*. Eingesehen am 20.03.2017 15:33 Uhr. URL: [http://www.obd-diagnose-shop.de/images/maxidas\\_ds708\\_ecutrade\(1\).jpg](http://www.obd-diagnose-shop.de/images/maxidas_ds708_ecutrade(1).jpg).
- [dsc] Daniel Schüller. *Bussysteme im Automobil*. Eingesehen am 13.03.2017 15:49 Uhr. URL: <https://userpages.uni-koblenz.de/~zoebel/ws2004/Bussysteme.pdf>.
- [emh] emotive GmbH. *K-Line-History*. Eingesehen am 29.03.2017 16:40 Uhr. URL: <https://www.emotive.de/images/stories/K-Line-History.png>.
- [emoa] emotive GmbH. *Diagnosesysteme im Automobil*. Eingesehen am 16.03.2017 09:41 Uhr. URL: <http://www.emotive.de/documents/WebcastsProtected/Transport-Diagnoseprotokolle.pdf>.
- [esd] Ilya Dinmukhametov Michael Elizarov Anton Sysoev. *How to find control-/signal CAN frames*. Eingesehen am 03.04.2017 08:59 Uhr. URL: [https://github.com/eik00d/CANToolz/wiki/find\\_frames](https://github.com/eik00d/CANToolz/wiki/find_frames).
- [eti] Equipment & Tool Institute. *OEM Licensing*. Eingesehen am 04.04.2017 08:24 Uhr. URL: <http://www.etools.org/OEMLicensing/>.
- [euo] Europäisches Parlament / Europäischer Rat. *Richtlinie 98/69/EG des europäischen Parlaments und des Rates vom 13. Oktober 1998 über Maßnahmen gegen die Verunreinigung der Luft durch Emissionen von Kraftfahrzeugen und zu Änderung der Richtlinie 70/220/EWG des Rates*. Eingesehen am 21.03.2017 16:33 Uhr. URL: [http://eur-lex.europa.eu/resource.html?uri=cellar:9ee5d16b-1a4a-4a72-ac90-5e3a0bb1d745.0004.02/DOC\\_1&format=PDF](http://eur-lex.europa.eu/resource.html?uri=cellar:9ee5d16b-1a4a-4a72-ac90-5e3a0bb1d745.0004.02/DOC_1&format=PDF).
- [evc] Eric Evenchick. *pyvit: Python Vehicle Interface Toolkit*. Eingesehen am 13.07.2017 14:46 Uhr. URL: <https://github.com/linklayer/pyvit>.
- [eve] Eric Evenchick. *CANTact – The Open Source Car Tool*. Eingesehen am 03.04.2017 17:28 Uhr. URL: <http://linklayer.github.io/cantact/>.
- [ewc] Elizabeth Weise (CNBC). *Chinese group hacks a Tesla for the second year in a row*. Eingesehen am 09.08.2017 08:48 Uhr. URL: <https://www.cnbc.com/2017/07/28/chinese-group-hacks-a-tesla-for-the-second-year-in-a-row.html>.

- [fdbp] Florian Hartwich (Robert Bosch GmbH). *CAN with Flexible Data-Rate*. Eingesehen am 29.03.2017 09:46 Uhr. URL: [http://www.bosch-semiconductors.de/media/ubk\\_semiconductors/pdf\\_1/ipmodules\\_1/can\\_fd/icc13\\_2012\\_paper\\_Hartwich.pdf](http://www.bosch-semiconductors.de/media/ubk_semiconductors/pdf_1/ipmodules_1/can_fd/icc13_2012_paper_Hartwich.pdf).
- [fdvb] Vector Informatik GmbH. *CAN-FD: Flexible Data Rate CAN – An Abbreviated Primer*. Eingesehen am 28.03.2017 16:13 Uhr. URL: [http://www.ukintpress-conferences.com/uploads/SPTXNA12/Day2\\_13\\_Rick\\_Lotoscky.pdf](http://www.ukintpress-conferences.com/uploads/SPTXNA12/Day2_13_Rick_Lotoscky.pdf).
- [fdvp] Vector Informatik GmbH. *Wege vom klassischen CAN zum verbesserten CAN FD*. Eingesehen am 28.03.2017 14:42 Uhr. URL: [https://vector.com/portal/medien/cmc/press/Vector/CAN-FD-ElektronikAutomotive\\_201304\\_PressArticle\\_DE.pdf](https://vector.com/portal/medien/cmc/press/Vector/CAN-FD-ElektronikAutomotive_201304_PressArticle_DE.pdf).
- [fdvs] Vector Informatik GmbH. *CAN FD – CAN with Flexible Data Rate – Vector Kongress 2012*. Eingesehen am 28.03.2017 15:28 Uhr. URL: [https://vector.com/portal/medien/cmc/events/commercial\\_events/VectorCongress\\_2012/VeCo12\\_8\\_NewBusSystems\\_3\\_Lindenkreuz\\_Lecture.pdf](https://vector.com/portal/medien/cmc/events/commercial_events/VectorCongress_2012/VeCo12_8_NewBusSystems_3_Lindenkreuz_Lecture.pdf).
- [fisa] Thomas Fischl. *USBtin EB USB-CAN Adapter Evaluation-Board SMD*. Eingesehen am 03.04.2017 17:22 Uhr. URL: [https://images-na.ssl-images-amazon.com/images/I/61pW-6Hz4IL.\\_SL1000\\_.jpg](https://images-na.ssl-images-amazon.com/images/I/61pW-6Hz4IL._SL1000_.jpg).
- [fisb] Thomas Fischl. *USBtin - USB to CAN interface*. Eingesehen am 03.04.2017 17:31 Uhr. URL: <http://www.fischl.de/usbtin/>.
- [gil] Python Community. *GlobalInterpreterLock*. Eingesehen am 18.07.2017 11:22 Uhr. URL: <https://wiki.python.org/moin/GlobalInterpreterLock>.
- [hjs] H.-J. Schleißheimer Soft- und Hardware Entwicklung GmbH. *K-Matrix oder CAN-Matrix*. Eingesehen am 15.03.2017 16:05 Uhr. URL: [http://www.schleissheimer.de/caneasyhilfe/index.html?k-matrix\\_oder\\_can-matrix.htm](http://www.schleissheimer.de/caneasyhilfe/index.html?k-matrix_oder_can-matrix.htm).
- [hwh] Godai Group LLC (Hackerwarehouse). *CANtact Bundle*. Eingesehen am 03.04.2017 17:20 Uhr. URL: <http://3i2kbu3p64zdryox2qp5q43na.wpengine.netdna-cdn.com/wp-content/uploads/2017/01/CANtact-454A9387a.jpg>.
- [hws] Hans-Werner Schaal (Vector Informatik GmbH). *Ethernet und IP im Kraftfahrzeug*. Eingesehen am 25.07.2017 10:24 Uhr. URL: [https://vector.com/portal/medien/cmc/press/PON/Ethernet\\_IP\\_ElektronikAutomotive\\_201204\\_PressArticle\\_DE.pdf](https://vector.com/portal/medien/cmc/press/PON/Ethernet_IP_ElektronikAutomotive_201204_PressArticle_DE.pdf).
- [hxr] Hex-Rays SA. *Products: Disassembler & Decompiler*. Eingesehen am 18.04.2017 09:22 Uhr. URL: <https://www.hex-rays.com/products.shtml>.

- [isg] Infotip Service GmbH. *Differentielle Signalübertragung*. Eingesehen am 23.03.2017 09:31 Uhr. URL: <https://kompendium.infotip.de/differentielle-signaluebertragung.html>.
- [ISO9141-2] International Organization for Standardization (ISO). *ISO 9141-2: Road vehicles – Diagnostic systems – Part 2: CARB requirements for interchange of digital information*.
- [jhca] Henrik Ferdinand Nölscher Javier Vazquez Vidal. *Code White's CANBadger*. Eingesehen am 04.04.2017 13:37 Uhr. URL: <https://github.io/CANBadger/>.
- [jhcb] Henrik Ferdinand Nölscher Javier Vazquez Vidal. *Hardwear.io 2016 :- Reverse Engineering vehicles Burpsuite Style by Javier & Ferdinand*. Eingesehen am 26.04.2017 08:47 Uhr. URL: [https://www.youtube.com/watch?v=\\_RJvZIog04&spfreload=10](https://www.youtube.com/watch?v=_RJvZIog04&spfreload=10).
- [jtc] Jared M. Smith. *Awesome Vehicle Security*. Eingesehen am 03.04.2017 17:07 Uhr. URL: <https://github.com/jaredthecoder/awesome-vehicle-security>.
- [jve] Javier Vazquez Vidal. *EDC16 Demo Desktop*. Eingesehen am 20.04.2017 07:54 Uhr. URL: [https://github.com/fjvva/ecu-tool/blob/master/Desktop%5C%20Firmware/EDC16\\_Demo/Desktop/EDC16\\_Demo/Desktop.ino](https://github.com/fjvva/ecu-tool/blob/master/Desktop%5C%20Firmware/EDC16_Demo/Desktop/EDC16_Demo/Desktop.ino).
- [ksc] Käfer-Club Obergrafendorf. *Käfer-Schaltpläne: 1972 Beetle and Super Beetle*. Eingesehen am 13.03.2017 16:02 Uhr. URL: <http://www.vw-kaeferclub.com/images/bug72.jpg>.
- [kun] KUNBUS GmbH. *Anforderung an Bussysteme im Fahrzeug*. Eingesehen am 14.03.2017 11:28 Uhr. URL: <https://www.kunbus.de/anforderung-an-bussysteme-im-fahrzeug.html>.
- [lem] emotive GmbH. *LIN - Local Interconnect Network*. Eingesehen am 29.03.2017 16:15 Uhr. URL: <https://webcache.googleusercontent.com/search?q=cache:PklLg2vp3kcJ:https://www.emotive.de/doc/car-diagnostic-systems/bus-systems/lin+&cd=3&hl=de&ct=clnk&gl=de&client=ubuntu>.
- [lis] Matthias Meier (TU Dortmund). *LIN Bus*. Eingesehen am 29.03.2017 12:58 Uhr. URL: <https://ess.cs.tu-dortmund.de/Teaching/PGs/autolab/seminarfolien/Meier-LIN.pdf>.
- [mes] ME-Meßsysteme GmbH. *CAN Bus Grundlagen*. Eingesehen am 23.03.2017 11:54 Uhr. URL: <https://www.me-systeme.de/de/support/grundlagen/can-bus-grundlagen>.
- [miva] Chris Valasek Dr. Charlie Miller. *Adventures in Automotive Networks and Control Units*. Eingesehen am 14.03.2017 15:31 Uhr. URL: <http://illmatics.com/content.zip>.

- [msb] Bundesamt für Sicherheit in der Informationstechnik. *Überblickspapier: Netzzugangskontrolle*. Eingesehen am 25.07.2017 11:28 Uhr. URL: [https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Grundschutz/Download/Ueberblickspapier\\_Netzzugangskontrolle.pdf?\\_\\_blob=publicationFile](https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Grundschutz/Download/Ueberblickspapier_Netzzugangskontrolle.pdf?__blob=publicationFile).
- [mvd] Chris Valasek Charlie Miller. *Adventures in Automotive Networks and Control Units (DEF CON 21)*. Eingesehen am 09.08.2017 09:08 Uhr. URL: <https://www.youtube.com/watch?v=n70hIu9lcYo>.
- [obda] Xoneca. *OBD-II type A female connector shape*. Eingesehen am 21.03.2017 16:52 Uhr. URL: [https://en.wikipedia.org/wiki/File:OBD-II\\_type\\_A\\_female\\_connector\\_shape.svg](https://en.wikipedia.org/wiki/File:OBD-II_type_A_female_connector_shape.svg).
- [obdb] Xoneca. *OBD-II type A female connector shape*. Eingesehen am 21.03.2017 16:53 Uhr. URL: [https://en.wikipedia.org/wiki/File:OBD-II\\_type\\_B\\_female\\_connector\\_shape.svg](https://en.wikipedia.org/wiki/File:OBD-II_type_B_female_connector_shape.svg).
- [obdm] SAE (Society of Automotive Engineers). *SAE J1979: E/E Diagnostic Test Modes*. Eingesehen am 22.03.2017 13:50 Uhr. URL: <https://law.resource.org/pub/us/cfr/ibr/005/sae.j1979.2002.pdf>.
- [obdt] Florian Schäffer. *LKW Adapter für 24 V*. Eingesehen am 22.03.2017 09:05 Uhr. URL: <https://www.blafusel.de/obd/lkw24v.html>.
- [oca] Fortag. *FORTAG OBD2 16Pin zu DB9 RS232 Serial Port Adapter Stecker Kabel OBDII*. Eingesehen am 04.04.2017 10:03 Uhr. URL: <https://www.amazon.de/FORTAG-16Pin-Serial-Adapter-Stecker/dp/B012QWHBZG>.
- [ocy] Top Auto Accessories Mall. *High Quality OBD2 Cable Extension Splitter 16 Pin Male to Dual Female Y OBD Cable*. Eingesehen am 04.04.2017 10:10 Uhr. URL: <https://www.aliexpress.com/item/High-Quality-OBD2-Cable-Extension-Splitter-16-Pin-Male-to-Dual-Female-Y-OBD-Cable-Free/32547079317.html?spm=2114.40010308.4.75.un0ofs>.
- [pbe] Patrick Beuth (Die Zeit). *Die Mutter aller Autohacks*. Eingesehen am 13.03.2017 09:14 Uhr. URL: <http://www.zeit.de/digital/datenschutz/2015-07/verkehr-auto-jEEP-hacker-rueckruf-folgen>.
- [psc] Patrick Schnabel. *Netzwerk-Topologie: Bus / Chain / Bus-Topologie*. Eingesehen am 14.03.2017 10:59 Uhr. URL: <http://www.elektronik-kompendium.de/sites/net/0503281.htm>.
- [psd] PySide Development Group. *PySide: Python for Qt*. Eingesehen am 13.07.2017 14:52 Uhr. URL: <http://wiki.qt.io/PySide>.
- [Smi16] Craig Smith. *The Car Hacker's Handbook – A Guide for the Penetration Tester*. San Francisco: no starch press, 2016. ISBN: 978-1-59327-703-1.
- [sph] Sphinx team. *Sphinx: Python Documentation Generator*. Eingesehen am 13.07.2017 13:40 Uhr. URL: <http://www.sphinx-doc.org/en/stable/>.
- [urh] Uli Ries (Heise). *Angriffswerkzeug Metasploit hackt jetzt auch Autos*. Eingesehen am 05.04.2017 09:23 Uhr. URL: <https://www.heise.de/security/meldung/Angriffswerkzeug-Metasploit-hackt-jetzt-auch-Autos-3626482.html>.

- [Wer14] Ralf Schmidgall Werner Zimmermann. *Bussysteme in der Fahrzeugtechnik: Protokolle, Standards und Softwarearchitektur* (5. Auflage). Springer Vieweg, 2014. ISBN: 978-3658024185.