Wolfgang Schöchl, BSc

Secure Software Remote Updates and Diagnosis for Vehicles

Master's Thesis

to achieve the university degree of
Master of Science

Master's degree programme: Computer Science

submitted to
Graz University of Technology

Supervisors
Ass.Prof. Dipl.-Ing. Dr.techn. Christian Steger
Dipl.-Ing. Dr.techn. Gerhard Griessnig (AVL List GmbH)

Institute for Technical Informatics

Faculty of Computer Science and Biomedical Engineering

Graz, October 2018

Statutory Declaration

I declare that I have authored this thesis independently, that I have not used other than the declared sources/resources, and that I have explicitly marked all material which has been quoted either literally or by content from the used sources.

Graz, _____ _____

Date Signature

Eidesstattliche Erklärung[1]

Ich erkläre an Eides statt, dass ich die vorliegende Arbeit selbstständig verfasst, andere als die angegebenen Quellen/Hilfsmittel nicht benutzt, und die den benutzten Quellen wörtlich und inhaltlich entnommenen Stellen als solche kenntlich gemacht habe.

Graz, am _____ _____

Datum Unterschrift

[1] Beschluss der Curricula-Kommission für Bachelor-, Master- und Diplomstudien vom 10.11.2008; Genehmigung des Senates am 1.12.2008

Abstract

Today's vehicles are increasingly connected and equipped with a growing number of electronic control units (ECUs). Unfortunately, this trend does not only provide benefits for both customers and vehicle manufacturers, but also introduces new, security-related concerns. As recent attacks on vehicle's security have shown, it is required to fix these security issues in a timely manner. Therefore, a framework for secure communication between vehicles and infrastructure is crucial.

In this thesis, we design and analyse a framework for secure communication between vehicles and infrastructure. Our proposed solution is designed for the use in the automotive industry where scalability and cost-effectiveness is required. To demonstrate the features of our framework, we discuss and implement concepts for deploying Over-the-Air updates and gathering telemetry data remotely. We then elaborate on the outcome of a security review and penetration test. Finally, we present the results of functional tests.

Keywords: ECU, OTA update, firmware, remote telemetry data, automotive security, connected car

Kurzfassung

Moderne Fahrzeuge sind mit einer wachsenden Anzahl an elektronischen Steuergeräten (ECUs) ausgestattet und verfügen zunehmend über eine Verbindung zum Internet. Diese Entwicklungen bergen nicht nur Vorteile für Kunden und Fahrzeughersteller, sondern werfen auch neue, sicherheitsbezogene Fragen auf. Angriffe auf die Kommunikationssysteme von Fahrzeugen und deren mögliche Auswirkungen zeigen, dass Sicherheitslücken möglichst schnell behoben werden müssen. Ein Konzept für sichere Kommunikation zwischen Fahrzeugen und Infrastruktur ist dafür Voraussetzung.

In dieser Arbeit wird ein Konzept für die sichere Kommunikation zwischen Fahrzeugen und Infrastruktur entworfen und analysiert. Das Konzept ist speziell für den Einsatz in der Automobilbranche, wo Skalierbarkeit und Kosteneffizienz hohe Priorität haben, ausgelegt. Um die Einsetzbarkeit dieses Konzepts zu demonstrieren werden Anwendungen für die sichere Übertragung neuer Software und für sichere Ferndiagnose über das Internet entwickelt. Anschließend werden die Ergebnisse einer Sicherheitsanalyse, eines Penetrationstests und der funktionalen Tests diskutiert.

Acknowledgements

I would like to thank my advisors Dr. Christian Steger from the Institute of Technical Informatics and Dr. Gerhard Griessnig from AVL without whom this thesis would not have been possible. I would particularly thank Dr. Christian Kreiner and Dr. Christian Hanser for their great support and excellent discussions.

I owe my deepest gratitude to my family and friends for helping and guiding me in all life situations. Thank you!

Graz, October 2018 Wolfgang Schöchl, BSc

Contents

Abstract iii

1 Introduction 1
- 1.1 Motivation . . . 1
- 1.2 Relevance of Cyber Security . . . 2
- 1.3 Examples of Practical Attacks . . . 3
  - 1.3.1 Chevrolet Volt and OnStar . . . 3
  - 1.3.2 Jeep Cherokee 2014 and Uconnect . . . 3
  - 1.3.3 Tesla Model S . . . 4
  - 1.3.4 BMW 320d Touring and ConnectedCar . . . 5
  - 1.3.5 Nissan Leaf and NissanConnect . . . 6
  - 1.3.6 Conclusion and Analysis of Practical Attacks . . . 6
- 1.4 Related Work . . . 7
  - 1.4.1 Uptane . . . 7
  - 1.4.2 EVITA . . . 10
  - 1.4.3 ARROWHEAD . . . 10
- 1.5 Connection to AVL . . . 11
- 1.6 Thesis Outline . . . 12

2 Preliminaries 13
- 2.1 Basic Principles of Cryptography . . . 13
  - 2.1.1 Symmetric Cryptography . . . 13
  - 2.1.2 Asymmetric Cryptography . . . 14
  - 2.1.3 Digital Signatures . . . 14
- 2.2 Transport Layer Security . . . 15
- 2.3 Cryptographic Message Syntax (CMS) . . . 15
- 2.4 Message Queue Telemetry Transport . . . 15
  - 2.4.1 The Publish-Subscribe Pattern in MQTT . . . 16
  - 2.4.2 Topics . . . 16
  - 2.4.3 Quality of Service . . . 16
  - 2.4.4 Security . . . 17
- 2.5 Vehicle Infrastructure . . . 18
  - 2.5.1 Electronic Control Unit . . . 18
  - 2.5.2 Protocols . . . 18

2.5.3 Database CAN (DBC) files 20

3 Security Analysis **22**
3.1 System Components 22
3.2 Assets 23
3.3 Attacker Model 24
3.3.1 Script Kiddies 25
3.3.2 Black Hat Hacker 26
3.3.3 White Hat Hacker 26
3.3.4 Tuning Scene 26
3.3.5 Thieves 27
3.3.6 Competitors 27
3.3.7 Attacker Model Characteristics Summary 27
3.3.8 Attacker Model for this Thesis 28
3.4 Threat Modeling 28
3.4.1 Microsoft STRIDE 29
3.4.2 Threats and Mitigations 30
3.4.3 Results 32

4 Design **34**
4.1 Components 34
4.1.1 Concentrator 34
4.1.2 AVL Device - SecureSmartHub 35
4.1.3 Backend 36
4.2 Secure Communication 36
4.2.1 Layer 1: Payload Encryption and Key Encapsulation 37
4.2.2 Layer 2: TLS 37
4.2.3 Layer 3: Concentrator Topic Permissions 38
4.2.4 Cryptographic Algorithms and Security Parameters 38
4.3 OTA Update 39
4.3.1 Package 39
4.3.2 Procedure 40
4.4 Remote Telemetry Data Gathering 42
4.4.1 Package 42
4.4.2 Procedure 42

5 Implementation **44**
5.1 Framework 44
5.1.1 XML Handler 45
5.1.2 ZIP Handler 45
5.1.3 CMS Handler 46
5.1.4 MQTT Handler 47

5.1.5 Controller 48
5.2 OTA Update 49
5.2.1 ECU Flashing Application 49
5.2.2 MQTT Topic Structure 51
5.2.3 Package Structure 52
5.2.4 XML File Structures 52
5.3 Remote Telemetry Data Gathering 54
5.3.1 Telemetry Data Gathering Application 54
5.3.2 MQTT Topic Structure 55
5.3.3 Package Structure 56
5.3.4 XML File Structure 57
5.3.5 Telemetry Data File Format 57

6 Evaluation and Results **59**
6.1 Hardware Setup 59
6.2 Security Audit and Penetration Test 59
6.3 Functionality Tests and Validation of Correctness 62
6.3.1 ECU Flashing 62
6.3.2 Telemetry Data Gathering 63
6.4 Results 65
6.4.1 Security Audit and Penetration Test 65
6.4.2 Functionality: OTA update 65
6.4.3 Functionality: Remote Telemetry Data Gathering 66

7 Conclusion and Future Work **68**
7.1 Conclusion 68
7.2 Future Work 69

Bibliography **77**

List of Figures

1.1 HSM deployment architecture designed and developed in EVITA [20]. 11
1.2 The mediator device with communication interfaces and basic building blocks developed within the ARROWHEAD project [11]. 12

3.1 System components. 23
3.2 DFD of the system's components modeled with the Microsoft Threat Modeling Tool. 30

4.1 The update request, deployment and install sequence. 41
4.2 The telemetry data gathering sequence. 43

5.1 The framework's software components. 44
5.2 Packing/unpacking multiple files into/from a single ZIP container. . 45
5.3 Architecture of the ECU flashing application. 49
5.4 ECU flashing procedure. The blocks highlighted in green do not require authorisation; the blocks highlighted in red require authorisation. . . 51
5.5 Architecture of the telemetry data gathering application. 55

6.1 The hardware setup. 60
6.2 Hardware setup for in-vehicle tests. 63
6.3 Visualisation of the vehicle's velocity [km/h] recorded over a timespan of 20 seconds. 67

1 Introduction

1.1 Motivation

The introduction of electronic components in the automotive industry is a high driver for innovation in this area. Modern vehicles are equipped with an increasing amount of Electronic Control Units (ECUs), which bring comfort, entertainment and, most importantly, safety to vehicles. Well known safety systems such as anti-lock braking (ABS), airbag or electronic stability control (ESC) rely on ECUs. Thus, correct functionality of these control units is crucial to passenger safety.

Nowadays, the software of a luxury car is highly complex with more than 100 million lines of code [37]. With the industry moving towards connected and self-driving cars, these numbers are steadily increasing. Although vehicle software is heavily reliable, it's not immune to software related bugs. According to a report from Stout Risius Ross [53], in the year 2015, 15% of 51 million recalls where software related. However, programming errors are not the only reason for software-related recalls.

In 2015 the United States Environmental Protection Agency (EPA) found that the German automaker `Volkswagen (VW)` intentionally manipulated cars' emissions. VW programmed diesel engine control units in such a way, that US limits on nitrogen oxides (NO_x) emissions were only adhered to during laboratory testing. VW stated that about 11 million cars worldwide were equipped with this so-called defeat device [43]. This global scandal is now known as Dieselgate and led to a recall of about 500,000 cars in the US and caused serious financial concerns for VW [7].

Whereas computers and smartphones are capable of Over-the-Air (OTA) updates, most vehicles lack such a feature. OTA updates allow for the addition of new features and general improvements of vehicles already in the field. Updating software remotely is a convenience for both the customer and the manufacturer: The customer does not have to care about workshop appointments, and the manufacturer can lessen its organisational software update efforts. Moreover, OTA updates can drastically reduce software-related recall costs.

This thesis aims to give insight into current research towards vehicle communication security. We analyse security-related failures from different manufacturers by looking at recent attacks on connected cars. To develop a secure OTA update architecture,

we analyse threats to a vehicle's communication infrastructure and define mitigation strategies. Based on these findings we design and develop a reference architecture for OTA updates and remote telemetry data gathering features. We evaluate the design by penetration testing the implementation.

1.2 Relevance of Cyber Security

Security is an integral part of modern communication infrastructure. However, nowadays almost no day passes without a critical security-related bug making the news. Especially the rise of the malware called WannaCry, that affected a broad range of Microsoft Windows versions, and its fast spread throughout the globe showed how critical and vulnerable today's systems and infrastructure are. The malware attack caused whole companies to shut down their business until all computers were cleaned and back up and running [9].

Another recent example of a massive attack on a critical internet infrastructure is the Distributed Denial of Service (DDoS) attack carried out by the Mirai botnet. The attack targeted the company Dyn and caused downtimes of popular websites including The Guardian, Netflix, Twitter, and Reddit. Mirai was mainly composed of IoT devices and casts a poor light on the security of such devices. This is especially relevant as vehicles are also considered to be part of the IoT ecosystem [3].

Whereas Microsoft reacted very quickly and pushed updates to all affected products, most IoT devices compromised by the Mirai bot remained unpatched. In today's thoroughly connected world it is, however, unacceptable to abandon device support and expose vulnerable devices to further attacks.

In conclusion, the above-mentioned attacks on today's infrastructure highlight the necessity for secure devices and fast updates. To further foster the need for secure updates, especially in the automotive industry, the following section presents real-world attacks on vehicles and their impact on vehicle and passenger safety.

1.3 Examples of Practical Attacks

1.3.1 Chevrolet Volt and OnStar

In 2015, security researcher Samy Kamkar [30] inspected a Chevrolet Volt for security vulnerabilities. The vehicle was equipped with a telematics module called OnStar. OnStar is developed by General Motors and offers various services such as unlocking doors, managing vehicle settings, Turn-by-Turn Navigation, or emergency response and vehicle anti-theft features. However, the security of the device and its communication with the vehicle owner's smartphone application were flawed and allowed for a Man-in-the-Middle attack. Specifically, the vulnerability Kamkar exploited was that the OnStar smartphone application failed to validate public key certificates correctly.

To demonstrate the vulnerability, Kamkar built a low-priced device, called OwnStar, which has to be planted near the target vehicle. When the owner connects to the vehicle, the connection is intercepted by the OwnStar device. OwnStar then acts as a proxy between the application and the vehicle and allows for eavesdropping on network traffic. This allowed Kamkar to extract user credentials from the application and lead to a full compromise of the vehicles OnStar system. Besides unlocking doors and flashing lights, Kamkar was also able to read the vehicle owner's emails and track the vehicle's location.

Later it was discovered that General Motors had already been aware of the problems concerning their OnStar system. Five years before Kamkar's attack, Koscher et al. [32] and Checkoway et al. [10] reported multiple vulnerabilities to General Motors but did not publicly disclose their findings. Since no update mechanism for the OnStar system was in place, General Motors did not act on the information given by the researchers, leaving 1.2 million vehicles open for attacks.

1.3.2 Jeep Cherokee 2014 and Uconnect

In contrast to the attack described in Section 1.3.1, where an attacker has to be in proximity to the target vehicle, Miller et al. [34] managed to compromise a car via cellular network.

A major goal of Miller et al. was to take full control of the vehicle. After analysing the architectures and attack surfaces of a wide range of vehicles, the researchers found the Jeep Cherokee 2014 to be the most vulnerable. Chrysler, the manufacturer of the Jeep Cherokee 2014, failed to separate the entertainment communication systems from the powertrain communication systems.

Similar to General Motor's OnStar, the Jeep Cherokee uses a system called Chrysler Uconnect. Uconnect handles infotainment, Wi-Fi, Bluetooth and cellular communication. It is also responsible for navigation and information retrieval, such as weather or traffic information. Due to its variety of features, such a system is highly complex and, thereby, has a broad attack surface. Hence, the researchers chose the Uconnect system as their main target.

Uconnect has the ability to update itself. For updating the Uconnect software, the update files are stored on a USB stick which is then plugged into the Uconnect USB port. After initiating the update process, the update files are checked for their validity and, if valid, the Uconnect system restarts and boots into update mode. In this mode, however, update files are not verified again. Due to this fact, an attacker is now able to modify update files and trick the system into installing a malicious software update. This flaw in the update process was the main entry point for Miller et al. and gave them full access to the Uconnect system. From there, the researchers started to explore and inspect other systems for possible bugs and misconfigurations.

Besides the aforementioned vulnerability, which requires physical access, the researchers managed to exploit the car via Wi-Fi. Multiple open ports, with one of them offering a remote procedure call (RPC) interface, where exposed via the wireless LAN interface. This allowed for easy command injection.

Although the Uconnect was fully compromised, it was not possible to directly write messages to the CAN bus. However, in order to gain full control over the vehicle, it is inevitable to write and read CAN messages. Miller et al. exploited another design flaw within the Open Multimedia Applications Platform (OMAP) chip which allowed them to rewrite the OMAP chip firmware with their own, modified firmware. This lead to the complete control over the vehicle, including brakes, throttle, and steering.

It should be noted, that the vulnerabilities described in this section were exploitable not only via Wi-Fi network but also via cellular communication. These attacks show that the manufacturer failed to implement even the most basic security mechanisms such as authentication or correct validation of software updates.

1.3.3 Tesla Model S

Tesla is known for their research on autonomous driving and connected cars in general. Tesla also provides a smartphone application and multiple in-car internet applications such as navigation or web browsing. A team of researchers [40] analysed a Tesla Model S, found several vulnerabilities, and exploited them to gain full access to the vehicle.

By setting up a rouge Wi-Fi access point, the Model S was tricked into automatically connecting to this fake access-point, allowing the researchers to inspect the web browser. It turned out that Tesla used an outdated web browser framework with known exploitable vulnerabilities. The successful exploitation of the web browser lead to a shell on the Model S entertainment system, revealing that the system was running Linux. The shell, however, was restricted and further steps were necessary.

By inspecting the Linux operating system, the researchers found that Tesla also used outdated versions of the Linux kernel, also with multiple known vulnerabilities and with almost no exploitation prevention techniques in place. The researchers used publicly available exploits to elevate their privileges to root. With full control over the embedded device, it was possible to write messages to the CAN bus and control safety-critical features such as ESP or ABS.

As all Tesla cars are already equipped with remote update features, most vulnerabilities were mitigated within ten days after the researchers reported the security flaws to the Model S manufacturer.

This research conducted by Nie et al. highlights the necessity for up-to-date software and the ability to update software components over-the-air. It should be noted that Tesla's response and the time needed to fix the reported vulnerabilities were very commendable.

1.3.4 BMW 320d Touring and ConnectedCar

On behalf of the Allgemeiner Deutscher Automobil-Club (ADAC), Spaar [52] conducted a research with the goal to get insight into what kind of data is transmitted to the manufacturers. Surprisingly, although the researcher focused on privacy concerns, he exposed multiple critical security flaws in BMW's ConnectedCar service. ConnectedCar is similar to services from other manufacturers and provides various remote management features via a smartphone app.

The researcher inspected the firmware of the module responsible for ConnectedCar services and exfiltrated multiple static encryption keys. Since these keys are identical for all BMW vehicles equipped with ConnectedCar, it was easy to encrypt and decrypt messages for other cars. Furthermore, BMW made use of the outdated and insecure encryption algorithm Data Encryption Standard (DES) and did not use transport encryption while communicating with the ConnectedCar cloud service.

Equipped with a laptop, Spaar managed to act as a ConnectedCar backend server by faking a cellular base station, and tricked multiple cars into unlocking doors.

BMW made several major mistakes in their ConnectedCar system. Static and identical keys stored in insecure storage, and usage of a known-to-be-broken encryption algorithm had serious impacts on the security of the ConnectedCar system.

1.3.5 Nissan Leaf and NissanConnect

The Nissan Leaf is one of the most sold electric cars on the market and comes with the NissanConnect system. In order to use NissanConnect, a subscription is required, and customers have to install the NissanConnect application on their smartphone.

The smartphone application allows customers to check the state of the battery, the remaining time until the battery is fully charged, or gives an estimate of the driving range and further suggests whether to switch the climate control system on or off.

All Nissan Leaf cars with an active NissanConnect subscription are connected to a cloud service provided by Nissan. The cloud service exposes an API which receives requests from the app and forwards them to the car. Listing 1.1 illustrates such an API request.

```
GET https://[redacted].com/orchestration_1111/gdc/
   BatteryStatusRecordsRequest.php?RegionCode=NE&lg=no-NO&
   DCMID=&VIN=SJNFAAZE0U60XXXXX&tz=Europe/Paris&TimeFrom
   =2014-09-27T09:15:21
```

Listing 1.1: A request to get the status of the climate control to the NissanConnect API [24].

When inspecting the request one can see, that the only unique identifier in the API request is the Vehicle Identification Number (VIN). Thus, no authentication mechanism is used to restrict access to the car owner only. According to Troy, the VIN of Nissan Leafs only differ in the last five digits and therefore allow for easy brute forcing [24]. A possible attack could be to deplete the batteries by switching on the climate control system, effectively rendering the car unusable until the owner recharges the batteries.

Although this attack is not as serious as the attacks described in Section 1.3.2 it shows, again, how often vehicle manufacturers fail to implement basic security mechanisms.

1.3.6 Conclusion and Analysis of Practical Attacks

Mistakes made by several manufacturers were missing or faulty certificate or update package validation. We can see from the above-mentioned attacks that it is crucial for

cryptographic implementations to correctly validate these certificates. Additionally, without proper update package verification, an attacker can effortlessly modify updates and trick systems into running malicious software. In general, the selection of cryptographic schemes is crucial for a system's security and usage of known-to-be-broken encryption algorithms is reckless.

Another failure was missing authentication to network services and APIs. If an important service is reachable via network, it should utilise authentication to prohibit access to unauthorised peers. Using publicly available information, e.g., the VIN, to authenticate peers is missing the point of authentication. An attacker may try to brute force VINs or, since most vehicles have the VIN printed on a sticker on the windshield, just read it from there. It should be noted that relying on flawed authentication may be worse than using no authentication at all, since it gives the impression of being secure when in fact it is not.

One manufacturer ran outdated software on its central communication and entertainment unit. Utilising software with known security flaws and publicly available exploits is a costly mistake and makes it easy for attackers to compromise the vehicle's security. An even more severe mistake is running an outdated operating system with no exploit mitigation in place. Modern up-to-date operating systems provide advanced defense mechanisms such as Non-executable stack (NX) [15] or Address Space Layout Randomization (ASLR) [54].

It is also advised against having multiple networked services exposed, i.e., open ports, to the entire world via cellular communication. This allows an attacker to remotely scan for software vulnerabilities and exploit them.

A full compromise of a vehicle is only possible if an attacker finds a way to write messages to powertrain or safety-critical bus systems. Physically separating entertainment and safety-critical bus systems may be one solution that would make it highly unlikely to get full control over a vehicle.

Table 1.1 lists design flaws and proposed mitigation techniques.

1.4 Related Work

1.4.1 Uptane

`Uptane`[1] [31] is a software update framework designed and developed by US-based universities, vehicle manufacturers and government regulators. Vehicle updates

[1] https://uptane.github.io

Design flaw	**Mitigation**
Missing or faulty certificate validation	Ensure proper validation of certificates
Missing or faulty update package verification	Ensure proper verification of update packages
Missing or faulty authentication mechanism	Ensure proper use and implementation of authentication mechanisms
Using outdated software	Use latest available software and check for known security bugs
Missing operating system defense mechanisms	Enable exploit mitigation techniques shipped with modern operating systems
Static keys	Distribute individual keys for all endpoints
Broken encryption schemes	Use state-of-the-art encryption schemes
Insecure storage	Use secure storage with tamper protection
Exposed network services	Keep exposed network services to a minimum
Interconnected safety-critical and entertainment bus systems	Physical separation of safety-critical and entertainment bus systems

Table 1.1: Faults and misconfigurations in connected car implementations and proposed mitigations.

greatly differ from PC update solutions: a) a vehicle has many different types of ECUs with different software versions or even different architectures and b) vehicle failure issues caused by an attacker are rarely managed. The `Uptane` framework was designed and developed with these aspects in mind and enhances already existing update designs by adding signed metadata.

Additionally, a comprehensive and broad threat model tailored to ECUs was developed. According to the authors and their threat model, an ECU should be immune to the attacks listed in Table 1.2.

Attack	**Description**
Eavesdrop attack	Gain information by reading unencrypted packets transmitted over networks
Drop-request attack	Hinder an ECU from receiving updates by blocking network traffic
Freeze attack	Hinder an ECU from updating to newer versions by always sending the same update
Partial bundle installation attack	Block updates from one or multiple ECUs by blocking network traffic, causing different installed software versions
Rollback attack	Install outdated software on an ECU
Endless data attack	Cause an ECU to run out of storage by sending data indefinitely
Mixed-bundles attack	Cause one or multiple ECUs to install different incompatible software versions and thereby hinder them to interoperate
Mix-and-match attack	Similar to the Mixed-bundles attack except that an attacker has gained access to private signing keys and can create its own update bundles
Arbitrary software attack	Update an ECU with an attacker developed software

Table 1.2: Attacks an ECU should be immune to [31].

1.4.2 EVITA

Secure in-vehicular communication is crucial to the success of connected cars. To ensure passenger safety, it is necessary to implement security features already at sensor level. All internal bus systems have to be secured against attacks in such a way that an attacker can not cause safety-critical systems to malfunction.

E-safety Vehicle Intrusion proTected Applications (EVITA) [20] was an EU project running from 2008 until 2011. EVITA aimed to secure in-vehicular communication. To secure on-board networks, EVITA had the following requirements:

- Security: Secure in-vehicular communication by protecting all electronic components and the network they are connected to.
- Real-time: Handle up to several thousand cryptographically secure messages per second.
- Cost-effective: Keep costs low by creating hardware tailored to its specific purpose.

Based on these requirements the project partners developed a framework for secure on-board communication consisting of hardware and software components. For secure storage of confidential data, e.g., keys and certificates, and for secure cryptographic operations, hardware security modules (HSMs) were developed. As one requirement of EVITA was to keep costs low, EVITA specifies three different classes of HSM modules. For high-performance Vehicle-to-Infrastructure communication the Full HSM was proposed. The medium HSM is for securing in-vehicular communication, whereas the Light HSM is used for securing sensors and actuators. For interacting and controlling the HSMs software and protocols were designed and implemented. The proposed solutions were verified using model-based verification tools. The final results and interface specifications have been made available to the public [20]. Figure 1.1 depicts an example of the EVITA HSM deployment architecture.

1.4.3 ARROWHEAD

ARROWHEAD[2] was an EU-funded research project with focus on smart production. The project's goal was to improve interoperability between devices used in smart production environments, especially devices used in maintenance and service tasks.

The outcome was a highly modular framework. Basic building blocks and core components with specified interfaces for component and building block interaction were defined. The framework was not designed with special industries in mind and,

[2] http://www.arrowhead.eu

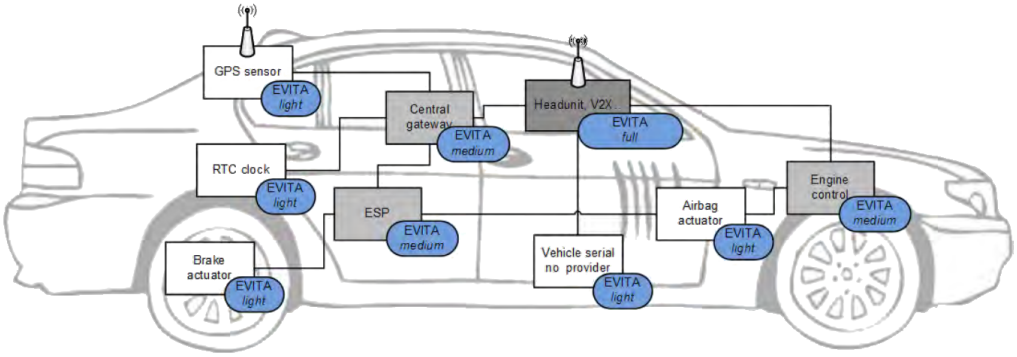

Figure 1.1: HSM deployment architecture designed and developed in EVITA [20].

therefore, enables devices from different industries to interact with each other. For example, energy production devices can communicate with home automation, process automation, and process monitoring devices. The information exchange between devices resulted in energy saving, efficiency improvement and cost reduction [11].

AVL[3] participated in the ARROWHEAD project. AVL's main objective was to bring secure connections to AVL products. ARROWHEAD project partners and AVL developed a central communication component called Mediator. Figure 1.2 depicts the layout of this central communication component.

The mediator was specifically designed for AVL's testing and measurement devices. However, since the ARROWHEAD framework applies to a wide range of applications and industries, it layd a good foundation for the concept developed within this thesis.

1.5 Connection to AVL

AVL List GmbH is an Austrian company with focus on research, design, and development of fuel-efficient, eco-friendly powertrains. AVL also develops test systems and advanced simulation technologies for testing and simulating powertrains. Additionally, AVL has considerable experience in developing safety-critical systems, especially in the automotive domain.

With the industry moving towards connected cars AVL has interest in acquiring know-how in this area, especially in the area of the rapidly growing cyber secu-

[3]http://www.avl.com

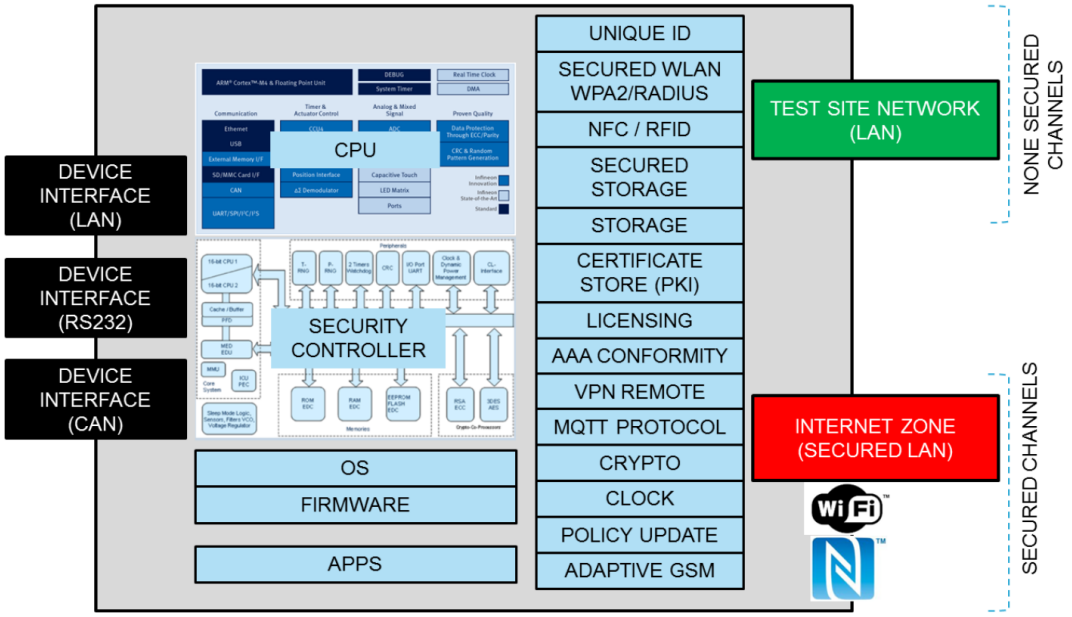

Figure 1.2: The mediator device with communication interfaces and basic building blocks developed within the ARROWHEAD project [11].

rity market. The challenges which arise with permanent connections to modern telecommunication infrastructures offer new business fields and chances for AVL.

In 2016 a cyber security department was founded and integrated into the safety department. To keep the role of a leading developer and researcher, AVL plans to expand its effort in the cyber security domain.

1.6 Thesis Outline

In the introduction we elaborated on the motivation behind this thesis. By illustrating previous, practical attacks the need for cyber security was further highlighted. The second chapter aims to explain basic elements of information security and vehicle infrastructure. In the third chapter we focus on security analysis and threat modeling. We elaborate on assets, attacker models, and design and analyse a framework for secure communication between vehicles and infrastructure. In Chapter 4 we build on the results of the security analysis and secure our framework against potential threats. Further, we use the secure communication framework for developing OTA update and remote telemetry data gathering features. Chapter 5 covers the implementation of a prototype. In Chapter 6 we evaluate the security and functionality of our implementation. In Chapter 7 we conclude and propose future research directions.

2 Preliminaries

2.1 Basic Principles of Cryptography

Cryptography allows for modifying data in such a way that an adversary can not deduce any useful information from the encrypted data. Encryption is the process of transforming a plaintext into ciphertext. Decryption is the inversion of the encryption process and produces plaintext from a given ciphertext.

Any cryptographic system must have at least one of the following properties:

- Confidentiality: Confidentiality was the main purpose for developing cryptographic systems. By fulfilling this property data is hidden from an eavesdropper and only the recipient with correct decryption instructions can extract valid and meaningful information.
- Authentication: Cryptographic systems with this property ensure that no one can impersonate, or send a message in the name of any other party. Authentication identifies a sender. Additionally, it proofs to the recipient that the message originated from the participant who claims to have sent the message.
- Integrity: Integrity ensures that encrypted data can not be tampered with without the receiver noticing it.
- Non-Repudiation: With non-repudiation a cryptographic system ensures linkability between an identity and a message. As soon as one sends data in a cryptographically secure manner he cannot deny the authenticity of the data he sent.

2.1.1 Symmetric Cryptography

In symmetric systems, exactly one key is used for encryption and decryption. A problem in symmetric systems is that the secret key has to be known by all participating parties before messages can be exchanged in a meaningful way. The security of symmetric encryption schemes is based on the confidentiality of the secret key. As soon as an adversary is in possession of the secret key he can encrypt and decrypt messages.

When people started to use symmetric cryptographic schemes (e.g. CAESAR, VIGENERE) a courier delivered the secret information necessary to decrypt ciphertexts to participating parties. However, the courier was subject to attacks and the key exchange dilemma remained unsolved until the introduction of asymmetric cryptography.

Well-known state-of-the-art symmetric encryption standards are the Advanced Encryption Standard (AES) and Triple Data Encryption Standard (Triple-DES). AES and Triple-DES are recommended by the National Institue of Standards and Technology (NIST) [1], [6].

2.1.2 Asymmetric Cryptography

In contrast to symmetric schemes, asymmetric schemes utilise multiple keys to achieve the properties mentioned in Section 2.1. Usually a key pair, consisting of a private and a public key, is generated. In order to keep the scheme secure the private key has to remain secret.

For example, if Alice wants to send an encrypted message to Bob, she encrypts the message with the public key of Bob. After Bob received the message, he uses his private key to decipher Alice's message.

Symmetric encryption schemes usually outperform asymmetric encryption with respect to speed. To combine the advantages of both cryptographical systems, so-called hybrid encryption is used. Hybrid encryption utilises an asymmetric encryption scheme to encrypt the secret key for a symmetric encryption scheme. Symmetric encryption is then used for further communication.

Recommended asymmetric encryption schemes are, for example, RSA and Diffie-Hellman [17].

2.1.3 Digital Signatures

Digital signatures are used to ensure integrity and authenticity of messages and are built on asymmetric cryptographical systems. The same properties as in Section 2.1.2 apply.

If Alice wants to sign a message, she calculates the signature with her private key and appends it to the message. Bob then uses Alice's public key to verify the integrity of the message and that Alice was indeed the sender of the message.

Recommended digital signatures are, for example, Digital Signature Algorithm (DSA) and Elliptic Curve-DSA (ECDSA) [29].

2.2 Transport Layer Security

Transport Layer Security (TLS) is the successor of Secure Sockets Layer (SSL) and is a cryptographical protocol which allows for secure communication between two parties. TLS provides data integrity, authentication and confidentiality and is widely used by a variety of applications, including web browsing and email. The TLS protocol operates on top of the TCP protocol.

During connection build-up, TLS uses a handshake protocol which allows the participating parties to agree on an authentication mechanism. Cryptographic primitives for traffic encryption are also selected in this phase.

Due to its widespread use, TLS was subject to multiple attacks in the past. Most notable attacks are POODLE [35], Lucky 13 [21] and BEAST [18]. Therefore, it is essential to TLS' security to use the latest TLS version with state-of-the-art cryptographic primitives and authentication mechanisms. The current version is TLS 1.2; TLS 1.3 is currently in the working draft state [16], [44].

2.3 Cryptographic Message Syntax (CMS)

Cryptographic Message Syntax (CMS), standardised in RFC 5652, is used for signing, encrypting and authenticating data. CMS can work on any data and allows, for example, to wrap unencrypted data into an encrypted envelop. This envelop can then be signed and authenticated and wrapped into another envelop. The envelop nesting level is not limited. Besides payload data, CMS also stores meta information such as used algorithms, serial numbers, or version information. This meta information is needed for correct handling of the payload data contained in a CMS message [22].

2.4 Message Queue Telemetry Transport

Message Queue Telemetry Transport (MQTT) is a protocol specifically designed for environments where network bandwidth is costly or only limited processing power and memory resources are available. When using the MQTT protocol only a small overhead for transport is added to the data. These properties render this protocol

ideal for the use in constrained environments like the Internet of Things [4]. In contrast to the commonly used client-server design principle, MQTT follows the publish-subscribe paradigm.

2.4.1 The Publish-Subscribe Pattern in MQTT

The publish-subscribe pattern is a principle for delivering messages where the sending client does not directly send messages to a receiving client. Instead, the sender publishes a message under a certain topic. Clients express interest in messages by subscribing to one or more topics. Only messages published to the subscribed topics are received by the clients. A client in the means of the publish-subscribe pattern can take the role of a publisher, a subscriber, or both. Besides the publisher and the subscriber, a third component is involved - the broker. The broker is known to all participating clients and is responsible for filtering and delivering messages. The publish-subscribe pattern has various advantages over the traditional client-server principle:

- Time: peers do not have to be online at the same time
- Space: peers do not have to know each other
- Synchronization: peers do not block each other while publishing/receiving
- Scalability: broker can be parallelized

2.4.2 Topics

Topics are strings following a specific level structure where each level is seperated by a slash ('/'). Topics are used by the MQTT broker for message filtering. For example, a humidity sensor, located in the kitchen, publishes humidity values to the `home/kitchen/humidity` topic. Another sensor responsible for opening and closing of windows may be interested in the humidity value in the kitchen and, therefore, subscribes to the corresponding topic. When a message gets published, the broker decides, based on the topic, to which clients the message gets delivered.

2.4.3 Quality of Service

MQTT offers three levels of Quality of Service (QoS). The QoS level defines how messages are delivered to clients. Table 2.1 shows available QoS levels.

QoS level	Description	Properties
0	at most once	• no guarantee that the message is delivered • messages are not acknowledged • messages may be lost • fastest transfer mode • should only be used when message loss is acceptable
1	at least once	• guarantee that messages are received by the client • duplicate messages may be received • should be used when receiving duplicate messages is acceptable
2	exactly once	• guarantee that messages are received exactly once • slowest transfer mode

Table 2.1: Available QoS levels in MQTT.

2.4.4 Security

Only authenticated clients should be allowed to subscribe and/or publish to MQTT topics and the content of messages must retain private. For this purpose, the MQTT protocol allows for traffic encryption and authentication mechanisms.

One method to authenticate a client is a username and password authentication mechanism. However, if MQTT traffic is not encrypted, username and password are sent in plaintext and are therefore subject to eavesdropping. Another method to authenticate clients is to utilise authentication features offered by the TLS protocol, e.g., client authentication. When using TLS, the content of published and received messages also remains private.

2.5 Vehicle Infrastructure

2.5.1 Electronic Control Unit

An Electronic Control Unit (ECU) is an embedded system and is responsible for controlling and governing in-vehicle systems. Such systems are, for example, the engine, the anti-lock braking system (ABS) or entertainment systems. Most ECUs follow the input-process-output (IPO) model. An ECU receives values from sensors distributed in the car. These values are then compared to calculated or previously defined setpoints. If the comparison results in a difference between the set point and the actual value, the ECU activates attached actuators to influence a physical process.

The size and complexity of an ECU varies and depends on its application. Although all ECUs are shipped pre-programmed, most of them can be updated by rewriting the ECU's internal flash memory. As reprogramming ECUs is prone to errors, only authorised personal is allowed to update an ECU. Faulty control units can lead to misbehaviour and, further, to serious injuries or fatal crashes.

2.5.2 Protocols

Controller Area Network (CAN)

CAN is a serial bus developed by Bosch and is standardised in ISO 11898 [26]. The bus system originated from the need by the automotive industry to reduce complex cable harness. Due to resilience to electrical interference and error recovering features, CAN is nowadays prevalent and used by multiple industries.

In contrast to other protocols such as USB, CAN is not a point-to-point protocol. Data transmitted over a CAN network is available to all participating nodes in the network. Nodes can decide on whether they are interested in the data. Data sent on a CAN bus are sent in messages with speeds up to 1 Mbit/sec. A message consists of 0 to 8 bytes of data. Each message is associated with an identifier. The identifier has a length of 11 bit in the Standard CAN and 29 bit in the Extended CAN. The CAN standard prohibits duplicate address identifiers on active messages originating from different source nodes. Identifiers are used for message filtering and message prioritisation.

ISO 15765-2 Transport Protocol (ISO-TP)

The ISO 15765-2 Transport Protocol [28] allows for sending messages exceeding the maximum CAN payload size of 8 bytes. An ISO-TP message consists of up to 4095 bytes of data. When transmitting data using the ISO-TP protocol, so-called ISO-TP frames are sent. These ISO-TP frames contain Protocol Information Bytes (PCI) to allow the recipient proper reassembly of ISO-TP messages. ISO 15765-2 specifies four different frame types, listed in Table 2.2.

The following two addressing schemes are supported by the ISO-TP protocol:

1. Extended Addressing: Used when CAN identifiers of nodes in the network are not unique. The addressing information is stored in a PCI byte of the ISO-TP frame.
2. Normal Addressing: Used when CAN identifiers of nodes in the network are unique. As no additional addressing information is necessary, the payload size increases by 1 byte when compared to Extended Addressing.

Depending on whether Extended Addressing or Normal Addressing is used, the maximum payload size of an ISO-TP frame is 6 and 7 bytes, respectively.

Frame Type	**Description**
Single Frame (SF)	Used when payload fits into a single ISO-TP frame.
First Frame (FF)	Used when payload size exceeds the maximum payload size of an ISO-TP frame. The message gets split into multiple frames, namely one First Frame and multiple Consecutive Frames.
Consecutive Frame (CF)	Consecutive Frames contain remaining payload bytes of a multi-frame ISO-TP message.
Flow Control Frame (FC)	A Flow Control Frame is sent by the recipient of the ISO-TP message to inform the sender of protocol specific settings such as time between two Consecutive Frames. The FC frame gets sent after the reception of a First Frame.

Table 2.2: The four different types of ISOTP frames.

Unified Diagnostic Services (UDS)

Unified Diagnostic Services (UDS) is a widely adopted protocol for diagnostics and maintenance of Control Units. UDS is specified in ISO 14229 [27] and is mainly used by the automotive industry. The payload size is not specified and is bounded by the underlying transport protocol. An UDS message is comprised of a Service-ID (SID), and service specific parameter and data bytes. A device which implements the UDS standard has to support a variety of commands. Table 2.3 lists some of the available commands.

Service	Service ID	Description
Diagnostic Session Control	0x10	Different modes of operation are defined in the UDS standard. A Diagnostic Session Control message allows to change the operating mode. Modes of operation mainly differ in the access rights. For example, the Programming Mode allows to upload and write data to the ECU's memory whereas the Default Session is not privileged to write memory.
Security Access	0x27	When switching from low privilege modes to higher privileged ones it is necessary to request higher access rights. For this purpose, the Security Access is used.
Read DTC Information	0x19	Used for reading fault memory entries.

Table 2.3: Basic UDS commands.

2.5.3 Database CAN (DBC) files

Most ECUs communicate via the on-board CAN network. In order to describe how information is encoded in CAN messages, different file formats where specified. The most used format in the automotive industry is the proprietary Database CAN (DBC)

format, which was specified by the company Vector[1]. Vector offers multiple tools and libraries to view and modify DBC files. The DBC format specification, however, is not publicly available. Nevertheless, open source implementations capable of reading and converting DBC files exist[2,3].

A CAN message contains one or more signals. Each signal corresponds to exactly one ECU provided value. The general syntax for signals following the DBC format is shown in Listing 2.1. Listing 2.2 shows a sample DBC entry of a single signal, in this case, the current vehicle speed, measured in km/h.

```
<object> <name> : <start bit>|<length>@<endiannes><
   signedness> (<factor>,<offset>) <range> "<unit>" <nodes>
```

Listing 2.1: Syntax of a signal following the DBC format.

```
SG_ HVCUVehSpd : 11|16@0+ (0.05625,0) [0|299.98] "km/h"  EMS
```

Listing 2.2: Current vehicle speed signal following the DBC format.

[1]https://vector.com
[2]https://github.com/julietkilo/CANBabel
[3]https://github.com/Polyconseil/libcanardbc

3 Security Analysis

Security engineering is ideally incorporated into early phases of the system design process. An important part of security engineering is threat modeling. Threat modeling helps to identify potential weaknesses and security flaws in a system's design. Moreover, threat modeling prevents significant system changes in later development phases.

The first step when conducting a security analysis is to define a basic model of the system. This is followed by identifying assets. Assets are targets of an attacker and essentially the reason why threats exist in the first place. After assets are specified it is necessary to define an attacker model that stipulates the capabilities and limitations of an attacker. Finally, the system can be analysed for possible threats, and mitigations can be proposed.

In this chapter, we define our system's components, assets, and attacker model. We elaborate on threat modeling in general and, in particular, explain the Microsoft STRIDE threat modeling methodology in detail. We then apply Microsoft STRIDE to our system, discuss identified threats and propose mitigations.

3.1 System Components

As mentioned in Chapter 1, the goal of this thesis is to develop a framework for secure communication between vehicles and backend infrastructure. It should be possible to deploy OTA updates to vehicles and initiate remote telemetry data gathering via the backend. It is noteworthy that the ARROWHEAD project had similar goals, but laid focus on measurement devices. We, however, build on ARROWHEAD's basic structure and adapt it to fit our needs. Our system is comprised of the components depicted in Figure 3.1.

1. **Vehicle:** The vehicle is the component where updates should be deployed to. In addition, it should be possible to gather vehicle telemetry data remotely. Such data could include current velocity, brake fluid pressure, or current motor engine speed.

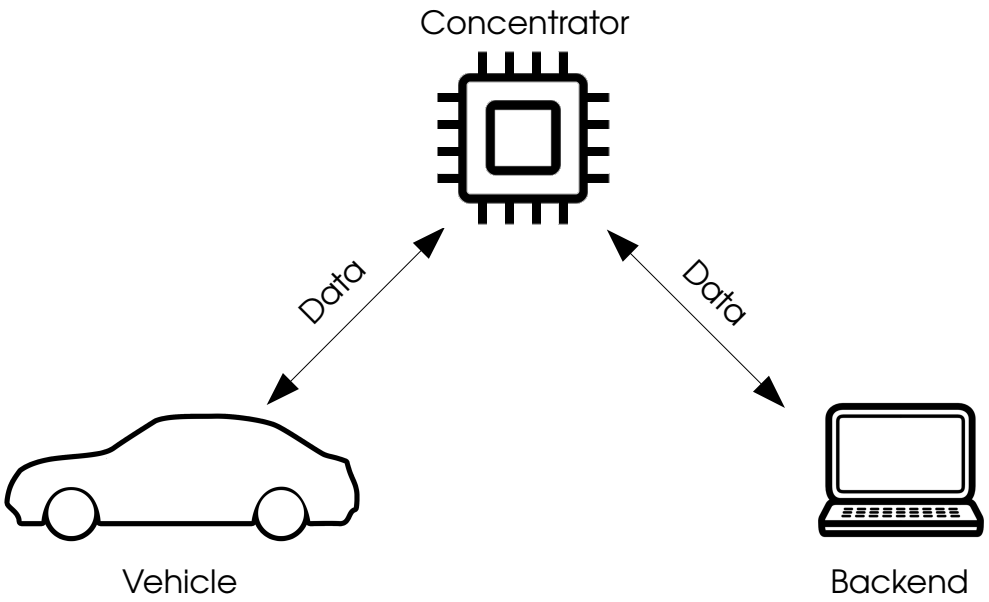

Figure 3.1: System components.

2. **Concentrator:** The concentrator is the backbone and central component of the system. All messages exchanged between backend and vehicle pass through the concentrator. This component handles and ensures successful message delivery.
3. **Backend:** The backend is the main component for user interaction and information visualisation. Deployment of updates and remote telemetry data gathering processes are initiated via this component. The backend stores, processes and displays all kinds of vehicle statistics, such as current ECU firmware versions, and provides fleet management functionalities.

3.2 Assets

An asset is a resource of value for both the asset owner and the attacker. Muckin et al. [38] divided assets into the following two categories:

1. **Physical:** Physical assets are data or devices of special interest to attackers. A database containing private information of a company's customers, private keys or certificates are types of physical assets. Physical assets are sometimes referred to as security assets.
2. **Abstract:** Abstract assets are objectives that are essential to a company's mission. Attackers who attack abstract assets influence a company's reputation.

For example, a company that acts as a service provider has to guarantee its customers that their service is available and operating correctly. An attacker, however, may try to bring down the service and thereby cause a negative impact on the company's reputation.

Hereafter, we define assets relevant for this thesis and categorise them according to [38]. Table 3.1 lists these assets.

Physical Assets

A highly valuable asset is data transmitted via wireless communication interfaces. Such data can contain firmware updates for ECUs, configuration parameters, DBC files specifying CAN message layout, or telemetry data. Successful attacks on the system or one of its components may result in extraction of confidential data. Whereas extraction of ECU or CAN related data affects the intellectual property, compromisation of telemetry data negatively impacts a drivers' privacy.

Besides the payload of transmitted packets, connection metadata is also a valuable asset to protect. Connection metadata may leak connection endpoints and requested resources. This information may be used for the generation of usage profiles.

Another important asset is the cryptographic key material stored on the vehicle. Successful extraction of private key certificates or symmetric encryption key material renders a secure communication concept useless. If an attacker is in possession of cryptographic keys, he is able to decrypt and encrypt data, impersonate vehicles, and send fake data to the backend.

Abstract Assets

The availability of systems is crucial for proper update deployment and remote telemetry data gathering. Whereas unstable systems can cause data loss, unavailable systems render all services useless. Since malfunctioning or misbehaving systems indirectly influence a company's reputation, system availability is a valuable asset.

3.3 Attacker Model

Attacker models specify capabilities of an attacker. Attackers' capabilities differ in, e.g., available resources, know-how, or time. Attackers are also often limited in the access to the device or service under test. In the following paragraphs, we elaborate

Asset	Category	Affected attributes
ECU firmware	Physical	Intellectual property
ECU configuration parameters	Physical	Intellectual property
DBC files	Physical	Intellectual property
Telemetry data	Physical	Privacy
Connection metadata	Physical	Intellectual property, Privacy
Cryptographic key material	Physical	Full system compromise
System availability	Abstract	Reputation

Table 3.1: Assets and affected attributes.

on various attacker models, as proposed by Spaan [51]. Based on these attacker models we specify the attacker model relevant for this thesis.

- **Resources:** Specifies what resources are available to attackers. This ranges from a simple personal computer with neglectable computation power to well-equipped laboratories, specialised tools, and software.
- **Know-how:** Describes how skilled attackers are. This ranges from simple attacks with available tools to highly sophisticated attacks with self-developed exploits.
- **Time:** Time is an important factor when attacking a system and is influenced by an attacker's motivation. For example, unskilled attackers stop attacking a system when tools and exploits do not lead to an immediate success.

3.3.1 Script Kiddies

Script Kiddies do not have a deep understanding of systems and lack both resources and knowledge to search for vulnerabilities and exploit them. Therefore, script kiddies use publicly available tools like `Metasploit`[1] to attack systems. These tools come pre-loaded with exploits for publicly known vulnerabilities.

Since Script Kiddies use existing tools and do not develop their own, the time invested into one specific target is low. If a tool does not lead to a successful attack in a short time frame a new target is selected. Attacks are carried out due to boredom or aimed to increase reputation amongst their friends or groups [46]. However, although Script Kiddies lack knowledge and resources, they can seriously harm a system.

[1] https://www.metasploit.com/

3.3.2 Black Hat Hacker

Black Hat hackers are highly skilled individuals with a deep understanding of system internals and extensive knowledge in the field of cyber security.

Resources available to Black Hat Hackers vary and range from a standard laptop to sophisticated infrastructure, including laboratories with expensive electrical equipment and high-performance servers. Black hat hackers are capable of finding and exploiting vulnerabilities on their own and devote a significant amount of time to this process [51].

Their motivation is to increase reputation in the hacking scene or destructive behavior against companies, groups or individuals. Moreover, unpatched vulnerabilities, so-called 0-day exploits or 0-days, can earn an immense amount of money when sold to 0-day acquisition companies or on the black market [47]. However, some black hats attack systems out of sheer fun (i.e. 'because they can').

3.3.3 White Hat Hacker

In contrast to black hat hackers, white hat hackers are considered to be 'good' hackers, without malicious intent. Their skill set and available resources are comparable to black hat hackers. When white hat hackers discover vulnerabilities, they report them to affected manufacturers and software vendors. White hat hackers can be hired to penetrate a system's security and, thereby, expose unknown vulnerabilities [51]. These hackers-for-hire are also sometimes referred to as penetration testers.

White hat hackers attack systems to increase reputation in the hacking scene or out of personal interest. Since more and more companies offer money for reported vulnerabilities, so-called bug bounty programs, money also plays a role in white hat hackers' motivation.

3.3.4 Tuning Scene

In general, tuners can be separated into two groups: 1) Tuners who reverse engineer the inner workings of vehicles, develop new configurations and software, and 2) Tuners who use available tools and configurations. These two groups significantly differ in knowledge, skills, and available resources. The motivation of both groups is driven by common goals: To increase engine performance, unlock premium features, or increase fuel efficiency. The first group also aims to increase their reputation by being the first to offer tuning features for new vehicles [51].

The market for customised vehicles is rapidly growing. Therefore, tuners try to earn money by selling their knowledge and tools to garages and individuals.

3.3.5 Thieves

Similar to tuners, thieves can be divided into two groups: 1) Highly skilled groups and individuals, often well organised and 2) Amateurs. The first group is comparable to black hat hackers by means of resources and know-how. They use their knowledge to build devices for defeating anti-theft features. The latter group is comparable to script kiddies: They lack a deep understanding of vehicle security and utilise available tools and exploits for breaking into vehicles to steal them [51].

Both groups share the same motivation: Earning money. The first group earns money selling equipment for breaking into cars and circumvent anti-theft technologies. These hard- and software components are bought and used by the latter group for stealing vehicles. The vehicles are then sold as a whole or torn apart in order to sell individual parts.

3.3.6 Competitors

A significant amount of time of the vehicle development process is consumed by adjusting and enhancing a vehicle's driving behavior and environmental properties, e.g., engine performance, steering behavior, or fuel consumption. These settings and know-how are highly valuable and must be protected from a manufacturers point of view. Competitors, however, may try to extract know-how and intellectual property and apply this gained knowledge to their vehicles and products [51].

Competitors have strong knowledge in vehicle development and vehicle security. Moreover, competitors have access to specialised tools and laboratories. Their motivation is to extract know-how and integrate findings into their vehicles and ecosystem.

3.3.7 Attacker Model Characteristics Summary

By analysing different adversaries, we have seen that they differ in available resources, know-how, time, and motivation. Whereas unskilled attackers do not impose severe risks, highly skilled and motivated adversaries can seriously harm a vehicle's security. Table 3.2 gives an overview of different attacker model characteristics.

Attacker	Resources	Know-how	Time	Motivation
Script-Kiddies	+	+	+	Reputation, Fun, Boredom
Black Hats	+/++	++/+++	++	Reputation, Money, Fun, Hatred
White Hats	+/++	+++	++	Reputation, Personal interest, Fun, Money
Tuners	+/++	+/++	+++	Reputation, Money, Fun
Thieves	+/+++	+/++	+/+++	Money
Competitors	+++	+++	+++	Extract intellectual property

Table 3.2: Characteristics of attacker models relevant for this thesis [51].

3.3.8 Attacker Model for this Thesis

One attacker property that was not discussed previously is physical access to the device under test. Hence, we add a physical access characteristic to our attacker model. This characteristic specifies whether an adversary has physical access to a device or not. The following enumeration lists our attacker's capabilities:

- Resources: Resources available to our adversary are comparable to black hat hackers and competitors. Sophisticated laboratories, tools, and software are available.
- Know-how: Our attacker is highly skilled with strong knowledge in the field of cyber security. In addition, our adversary has a deep understanding of vehicle infrastructure and internals.
- Time: The attacker is bounded by a time frame of a maximum of one year.
- Motivation: Our adversary combines all aforementioned motivations.
- Access: The attacker does not have physical access to the vehicle. All attacks have to be carried out via wireless or in-vehicle communication interfaces.

3.4 Threat Modeling

Threat modeling is a thoroughly researched topic and still an active field of research [13], [39], [49], [50], [55]. A widely known and accepted threat modeling

methodology is Microsoft STRIDE. Microsoft STRIDE was an integral part of the security analysis conducted in this thesis.

3.4.1 Microsoft STRIDE

STRIDE (**S**poofing, **T**ampering, **R**epudiation, **I**nformation Disclosure, **D**enial of Service, **E**levation of Privilege) was developed by Microsoft and is a model for classifying threats into six categories. Table 3.3 lists these threat categories.

Threat	**Description**	**Example attack**
Spoofing	Accessing data or a system by impersonating another user or system.	ARP spoofing
Tampering	Modifying data or a system without being authorised to do so.	Man-in-the-Middle attack
Repudiation	Denying that a certain action was performed.	Logfile manipulation
Information Disclosure	Leaking private information to third parties.	Man-in-the-Middle attack
Denial of Service	Rendering a service unusable.	Exploiting a bug to cause application crashes
Elevation of Privilege	Elevating access rights to perform privileged actions	Exploiting a bug to gain higher access rights

Table 3.3: STRIDE threat categories and example attacks.

One way to extract and categorise potential threats according to STRIDE is to use data flow diagrams (DFDs). A DFD models a system's components and how data traverses through a system and between the system's components [2]. Once a DFD is defined, threats can be extracted by inspecting the components and processes involved in a specific data flow.

To ease the process of applying the STRIDE threat modeling methodology, Microsoft provides the freely available Microsoft Threat Modeling Tool[2]. This tool allows to model systems by drawing DFDs and ships with templates for standard components, e.g., databases or web services. In addition, the Microsoft Threat Modeling Tool also provides features for defining custom and complex components. However, one major feature of the Microsoft Threat Modeling Tool is the generation of a comprehensive list of threats and possible mitigation strategies based on a DFD. The DFD of our system, modeled with Microsoft's threat modeling tool, is depicted in Figure 3.2.

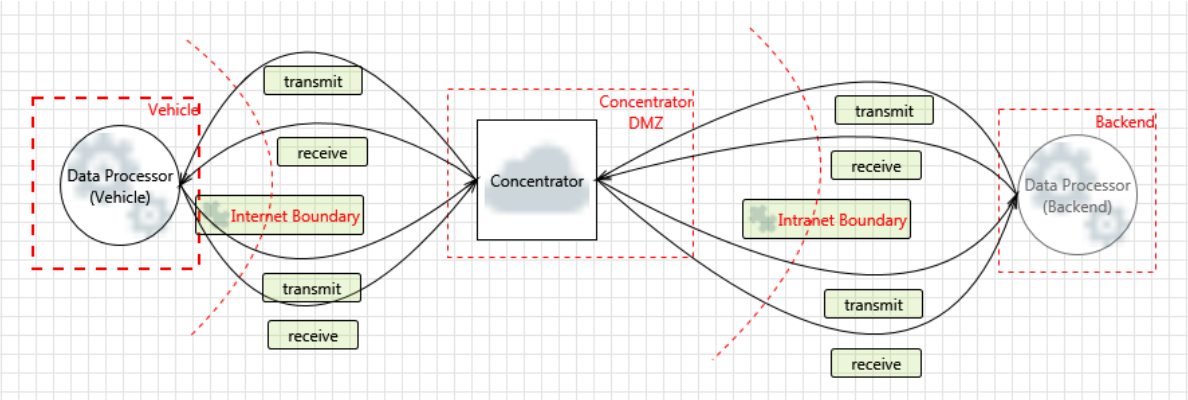

Figure 3.2: DFD of the system's components modeled with the Microsoft Threat Modeling Tool.

3.4.2 Threats and Mitigations

Microsoft's Threat Modeling Tool identified 26 potential threats. Listing all 26 threats and the discussion thereof would go beyond the constraints of this thesis. We therefore focus on the discussion of threat categories in the context of our system and propose mitigation strategies. A complete list of threats can be found in the Appendix.

Spoofing

An attacker may spoof one or more system components and, thereby, may be able to mount a Man-in-the-Middle attack. Such an attack, if carried out successfully, results in a complete breach of the system's security guarantees.

To counteract spoofing attacks, we incorporate strong authentication mechanisms into all components. Before payload data is transmitted, vehicles, the backend, and the concentrator have to authenticate themselves and proof their identity to the communication partner.

[2] https://www.microsoft.com/en-us/sdl/adopt/threatmodeling.aspx

Denial of Service

For example, if an attacker intercepts traffic and does not forward it to the correct entity, an attacker effectively interrupts the data flow between entities and causes the system to fail, i.e., Denial of Service. A Denial of Service attack can also be mounted by changing the data in transit and causing system components to malfunction or crash.

Mitigating Denial of Service attacks caused by communication interruption or any other form of hindering entities to communicate with each other, for example by overloading the concentrator with connection attempts, are out of scope for this thesis. However, an attacker should not be able to cause Denial of Service by injecting incorrect or malicious data. Therefore, our system must provide strong authentication mechanisms. All participating parties have to authenticate themselves, and all data has to be signed. A component receiving signed data has to validate the signature before considering data as trusted.

Elevation of Privilege

Elevation of Privilege attacks can be achieved by impersonating a participating entity. For example, an attacker may be able to impersonate the concentrator and send data to vehicles or the backend and, thereby, elevate its privileges. Another important attack often used for elevating privileges is remote code execution. If an attacker manages to control the program flow of programs running on one of the system components, the attacker may be able to elevate its privileges by executing code that normally should not be executed.

To eliminate Elevation of Privilege attacks, our system must ensure that only authenticated network participants can communicate with each other. This can be achieved by implementing strong authentication mechanisms. Additionally, all components must apply the principle of least privileges. For counteracting remote code execution attacks, all components must incorporate modern exploit mitigation and hardening techniques.

Repudiation

Without proper logging, vehicles, the concentrator, or the backend could claim that they did not receive data from other entities. This is especially interesting in the case of transmitting updates to vehicles where a vehicle could claim that it did not receive any update packages from the backend.

To mitigate Repudiation threats, our system must provide logging and sophisticated digital signature mechanisms.

Information Disclosure

An attacker may be able to inspect data flowing across communication channels and may access confidential and private information. By inspecting private information, an attacker could harm a vehicle owner's privacy by, for example, extracting the vehicle's position. Furthermore, through data leakage, an attacker also gains information about system internals that may lead to further attacks on the system.

Information Disclosure attacks are mitigated by incorporating strong encryption mechanisms into all system components. All data must be encrypted before leaving a device or trust boundary. We, therefore, must ensure that all entities use secure and widely-accepted encryption algorithms.

Tampering

Tampering causes participating entities to operate on wrong data and is a root cause of successful exploitations of all kinds of systems. For example, an attacker could modify data in such a way that the backend is tricked into believing that the firmware version of a vehicle is up-to-date, even though this is not the case. This would result in vehicles never receiving firmware updates. Furthermore, unauthenticated modification of data may lead to Denial of Service or Elevation of Privilege attacks.

In order to reduce Tampering threats, we must ensure that all data transmitted is authenticated. Moreover, all system components must validate data and incorporate plausibility checks before operating on data.

3.4.3 Results

Spoofing, Denial of Service and Elevation of Privilege are the categories with most threats identified. These three categories count for 18 of 26 threats. Repudiation counts for 4 threats whereas Information Disclosure and Tampering count for 2 threats each. Tampering and Information Disclosure threats can be mitigated by incorporating authenticated encryption. To allay Spoofing threats, all participating parties have to authenticate themselves. Extensive Logging on the vehicle, the concentrator, and the backend counteracts repudiation threats. By utilising authentication and authenticated encryption, some of the Denial of Service threats are mitigated. However, defending against Denial of Service attacks on critical communication infrastructure, especially

large-scale attacks, is an active field of research and not a trivial task. Due to the sheer amount of resources required for effective large-scale Denial of Service attack mitigation, these kind of attacks are out of scope for this thesis. Table 3.4 lists the number of threats identified.

Category	**#**	**# Out of scope**	**# Mitigated**
Spoofing	6	0	6
Tampering	2	0	2
Repudiation	4	0	4
Information Disclosure	2	0	2
Denial of Service	6	6	0
Elevation of Privilege	6	0	6
	26	6	20

Table 3.4: Threats categorised according to STRIDE.

4 Design

The concept developed within this thesis builds upon the framework developed in the ARROWHEAD project (see Section 1.4.3). Within the ARROWHEAD project, AVL's development focused on testing and measurement devices and, thus, does not fit our needs. As vehicles and measurement devices differ in nature, we incorporated significant changes into the ARROWHEAD framework.

In the following sections we propose a concept that allows for secure communication and data exchange between vehicles and infrastructure. We then elaborate on how this concept benefits the two primary use cases, OTA update and remote telemetry data gathering.

4.1 Components

The system is comprised of three components: 1) a vehicle that houses ECUs and the device responsible for handling communication with the concentrator, 2) the backend, where the system is controlled from, and 3) the concentrator that acts as a mediator between vehicles and the backend.

4.1.1 Concentrator

The concentrator is the central component of our communication concept. It is responsible for the routing of all messages exchanged between vehicles and the backend. As the number of vehicles communicating with the backend via the concentrator is going to grow in the future, the concentrator will be under heavy load when many messages are exchanged concurrently. Therefore, it should easily be possible to adapt and scale the concentrator component based on network traffic and concentrator load. Additionally, the concentrator should provide mechanisms for redundancy and fail-over in the case of technical problems. This ensures high availability and reliability.

To further lower the load on the concentrator, the interaction between vehicles and backend should be kept to a minimum. Moreover, to also keep the amount of

transmitted data to a minimum, the protocol used by the concentrator should have only small communication overhead. The concentrator should also be resilient against communication failures and guarantee successful message delivery to all peers. The message delivery guarantee also has to hold when not all peers are online at the same time. For example, the communication device incorporated into a vehicle may not be able to establish a connection to the concentrator when the vehicle is parked in a garage, or the vehicle is operated in areas with bad cell reception.

All data exchanged between vehicles and backend moves through the concentrator. This renders the concentrator an attractive target for attackers. An attacker that gains access to the concentrator or manages to tamper with data would have severe impacts on the system's security. In fact, this would result in a complete breach of incorporated security mechanisms. Vehicles must be able to communicate with the concentrator and the backend securely. Furthermore, before a message can be transmitted via the concentrator, all peers must authenticate themselves. Thus, the concentrator must provide mechanisms for authentication, encryption, and authorisation.

Ideally, the protocol used by the concentrator was designed with the aforementioned properties and requirements in mind. A protocol fulfilling our requirements is the MQTT protocol. MQTT provides features for authorisation and authentication. Additionally, MQTT was specifically designed for the use in the field of IoT where low-bandwidth capabilities and resilience against network failures are often a requirement. The QoS levels defined by MQTT allow for fine-grained settings on message delivery. To summarise, the MQTT protocol perfectly fits our needs.

4.1.2 AVL Device - SecureSmartHub

The device that handles communication between concentrator and vehicle, further AVL Device, should be capable of talking the MQTT protocol. Additionally, the AVL Device has to handle authentication, authorisation and encryption/decryption properly. It is, therefore, necessary to provide adequate hardware for processing complex cryptographic operations.

To communicate with other devices built into a vehicle, e.g., ECUs, the AVL Device must provide the corresponding interfaces. Modern cars come with a variety of bus systems. The AVL Device must, therefore, also be able to handle different bus systems and talk the protocols used on these bus systems.

Since AVL devices are integrated into vehicles where attackers have direct access to, we highly recommend to not only secure the communication between vehicle and backend but also to harden the AVL Device itself. For example, one could think of

using secure hardware elements for storing cryptographic key material and executing cryptographic calculations, or adding Secure Boot or Trusted Boot features.

4.1.3 Backend

Whereas the AVL Device is accessible from the vehicle owners, the backend is located within the company responsible for the management of AVL Devices. The backend is the component where vehicles are managed, and all information is stored. To allow for easy administration, the backend should provide a user interface. Via this user interface, the backend operator can manage and trigger OTA updates or initiate remote telemetry data gathering. Moreover, all data gathered during telemetry reading or ECU updating is to be visualised for further inspection and troubleshooting. The backend is also responsible for the management of cryptographic key material and should provide interfaces for the handling thereof.

Given the fact that the backend plays such an important role within our concept it is a high-risk component and an attractive target for hacking attempts. Therefore, the security of this component is crucial to the overall system's security.

From a technical point of view, the backend also has to be able to communicate with the concentrator via the MQTT protocol. Due to the massive load the backend will encounter it should be designed with scalability in mind. Additionally, since cryptographic key material and certificates are highly confidential, the backend must provide features for securely storing such data. Backend security was further analysed and discussed by Celina [8].

4.2 Secure Communication

End-to-end and confidential data transmission is achieved by layering three mechanisms. The first layer ensures the protection of payload data by encoding it into a CMS packet. To prevent connection meta data leakage, e.g., leaking topics a vehicle publishes or subscribes to, we incorporate a second layer: TLS. TLS ensures that vehicle-to-concentrator and backend-to-concentrator communication is encrypted and no confidential information is revealed. As a third layer, we incorporate fine-grained topic-specific permissions on the concentrator. These permissions are checked once a vehicle tries to subscribe or publish to a topic. The three layers of defense are explained in detail hereafter.

4.2.1 Layer 1: Payload Encryption and Key Encapsulation

The following paragraph elaborates on the process of sending a packet from a vehicle to the backend. For the reverse process, the keys have to be swapped accordingly.

When sending operational data to the backend, the AVL Device first has to generate an ephemeral key e_k. This key e_k is used for encrypting the payload according to the Advanced Encryption Standard (AES). As AES mode of operation we use the Galois/Counter Mode (AES-GCM) [19]. AES-GCM is a widely-adopted, authenticated encryption algorithm that guarantees both data integrity and confidentiality. Moreover, AES-GCM is very efficient and native hardware support for encryption and decryption operations is available in virtually all modern processors.

To wrap the symmetric key e_k, an ephemeral wrapping key w_{ek} has to be derived. We therefor use the Elliptic Curve One-Pass MQV (ECMQV) algorithm [5]. ECMQV is based on the Diffie-Hellman [17] key exchange scheme and allows for authenticated key agreement. The wrapping key w_{ek} is derived from the AVL Device-specific private key a_{sk} and the backend's public b_{pk}. The key w_{ek} is then used to wrap the key e_k according to the CMS key encapsulation mechanism (CMS KEM) [42]. To wrap the wrapping key w_{ek} for transport, we use the CMS `Authenticated-Enveloped-Data` content type [23].

To conclude, a CMS packet is comprised of 1) the encrypted payload, 2) the wrapped AES-GCM key, and 3) the ECMQV wrapping key. This CMS packet is sent to the concentrator according to the MQTT protocol. To retrieve the encrypted data, the recipient, i.e., the backend, has to execute the operations in reverse. First, the AES-GCM key e_k is unwrapped by using the wrapping key e_{wk} and the backend's private key b_{sk}. The key e_k is then used for decryption of the encrypted payload.

4.2.2 Layer 2: TLS

The use of TLS in conjunction with MQTT was discussed by Lesjak et al. [33]. In the following paragraph, we shortly present their findings and highlight the advantages of TLS in an MQTT scenario.

If messages are sent/received to/from the concentrator without utilising transport encryption, it is possible for an adversary to derive confidential information, e.g., the MQTT topic. Although an attacker can not read the payload of the MQTT message, this still imposes a security risk. This threat can be mitigated by utilising TLS for communication encryption. TLS does not only allow for traffic encryption and server authentication but also provides the functionality for authenticating clients against a server using X.509 certificates [12]. Using X.509 certificates in our system bears the

advantage that AVL Devices and the backend can authenticate themselves against the concentrator without the need for dedicated authentication mechanisms. TLS client authentication also reduces the attack surface of the concentrator by rejecting unauthenticated connection requests.

As proposed by Lesiak et al. we also use TLS 1.2. TLS 1.2 is the latest available version released in 2008 and mitigates the BEAST attack. However, TLS 1.3 is already in development, and one should think about upgrading to this version once it is released.

4.2.3 Layer 3: Concentrator Topic Permissions

As a third layer of defense, we apply fine-grained MQTT topic permissions. This ensures that only vehicles with proper access rights are allowed to subscribe or publish to specific topics. For example, it cannot be possible for vehicles from manufacturer `A` to subscribe or publish to topics from manufacturer `B` and vice versa. Although unauthorised recipients would not be able to decrypt packets not meant for them, it is possible to derive potentially confidential information, such as the number of vehicles a manufacturer operates. Additionally, this mechanism enhances the overall system security.

Since we already employ certificates for secure communication, it does make sense to also use them for incorporating topic permissions. We, therefore, utilise the `Common Name` (CN) field in X.509 certificates. The CN field specifies the topic this certificate is valid for and, thus, the topics a vehicle has permissions to. In the case that a vehicle has to have access to more than one topic, we use the `Subject Alternative Name` (SAN) extension for specifying additional topics. Certificates and topic permissions are checked for validity on the concentrator. The concentrator must reject publish/-subscribe attempts to topics that are not defined in the certificate provided by the AVL Device.

4.2.4 Cryptographic Algorithms and Security Parameters

For Authenticated Encryption we use AES-GCM with a key size of 256 bits. The Elliptic Curve Digital Signature Algorithm [29] is used for the creation of digital signatures. ECMQV, more specifically `One-Pass MQV, C(1e, 2s, ECC MQV)`, is used for authenticated key transport. For both elliptic curve algorithms, the `secp512r1` elliptic curve is used. We use the Secure Hashing Algorithm 2 with a key size of 512 bits (SHA-512) [14] where applicable. Since TLS is not standardised for SHA-512, we use SHA 2 with a key size of 384 bits (SHA-384). This results in the TLS cipher suite

string `TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384` [45]. Table 4.1 summarises used algorithms and key sizes.

Algorithm	Key size	Application
AES-GCM	256	Authenticated Encryption
ECDSA	521	Digital Signatures
ECMQV	521	Authenticated Key Exchange
SHA-2	384	Hashing (TLS)
SHA-2	512	Hashing (Non-TLS)

Table 4.1: Cryptographic Algorithms and Security Parameters.

4.3 OTA Update

In general, an ECU is shipped with software already pre-installed. As discussed in previous sections, however, it may be necessary to update this software. Therefore, update files and ECU-specific updating tools are required. Whereas the ECU vendor provides the updating tools, ECU updates are usually developed by the vehicle manufacturer.

Nowadays, when updates for an ECU are available, the vehicle manufacturer has to call all vehicles containing the ECU in question into a workshop. The vehicle mechanic then updates each ECU manually. This is costly and time-consuming. In the following paragraphs, we propose a scheme for deploying updates Over-the-Air for updating ECUs without the need for workshops and workshop personnel.

4.3.1 Package

Once update files are ready for deployment, they are packed into an update archive. As proposed by Karthik et al. [31] we also make use of a meta-information file. This meta-information file is packed amongst the update files into the update archive. The metafile contains information about update files and the update procedure. For example, to validate the integrity of update files, checksums are calculated and stored within the meta-information on update package creation. The integrity of update files can then be checked by the AVL device by calculating the checksums and comparing them against the checksums contained within the meta-information file. It is also possible that update files are depended on each other. Therefore, an `order` parameter, defining the order of how updates should be applied, is stored in

the meta-information file. However, the information stored in the meta-information file is vendor dependent and may be adjusted to fit the vendor's needs.

4.3.2 Procedure

When the vehicle initiates the update procedure, it first subscribes to the `Update Response` topic. It is crucial that the vehicle subscribes to the topic before the update request message is published. If this is not the case, messages may get lost between the publishing of the update request message and the subscription to the update response topic. As soon as the vehicle subscribed to the update response topic, the vehicle publishes an update request message to the `Update Request` topic. This message is received and processed by the backend. The backend checks for available updates, generates a meta-information file and creates an update archive. This update archive is then published by the backend to the `Update Response` topic. The vehicle receives the update archive, extracts it, and parses the meta-information file.

Depending on whether updates are available, the AVL Device behaves differently. In the case that no updates are available, the update procedure stops and no further action is taken. In the case that updates are available, the update handler validates the update files and starts the ECU flashing process. After all update files were processed, the update handler gathers the log files generated during the update process and packs them into an update report archive. This report archive gets published to the `Update Report` topic. The backend receives the update report archive, extracts it and stores the log files in a database. The backend operator is then able to inspect the log files. Figure 4.1 depicts the update procedure.

It should be noted that we do not take any safety measures into account. Deploying ECU updates without considering safety could result in an ECU update attempt while the vehicle is being operated. This may lead to vehicle misbehavior and fatal crashes.

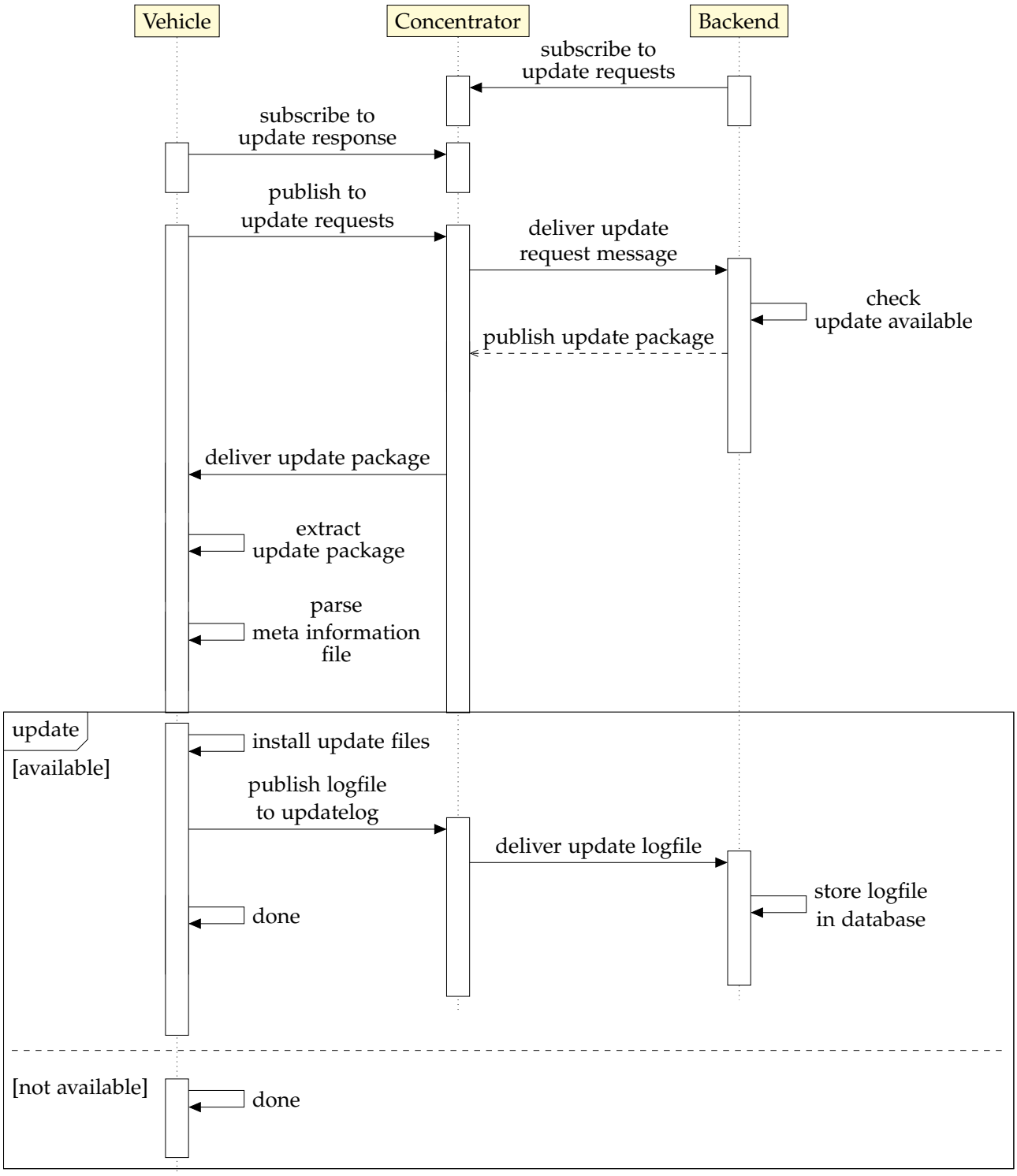

Figure 4.1: The update request, deployment and install sequence.

4.4 Remote Telemetry Data Gathering

Gathering telemetry data is an integral part of the vehicle development process. Engineers inspect collected data and, based on this data, adapt vehicle configuration parameters to improve vehicle performance. Vehicle mechanics also record and examine telemetry data for vehicle diagnostic and to locate errors. However, nowadays engineers or vehicle mechanics have to have physical access to vehicles that should be inspected. This is impractical and costly. In the following paragraphs, we propose a scheme for remote telemetry data gathering.

4.4.1 Package

Before a remote telemetry data gathering procedure can be initiated, the backend operator has to define the parameters and signals that should be gathered. These parameters and signals are stored in a file and packed amongst a meta-information file into an archive. Similar to the OTA update use case, the meta-information file specifies various parameters, e.g., the duration of the telemetry data gathering process.

4.4.2 Procedure

Initially, when a vehicle is started, the vehicle component subscribes to the `Telemetry Request` topic. When the backend operator initiates remote telemetry data gathering the backend publishes an archive containing the meta-information file and parameter description file to the `Telemetry Request` topic. On reception of the archive by the vehicle component, the archive is extracted. The archive contents are then parsed and the telemetry data gathering process is initiated.

While telemetry data is gathered, the vehicle component continuously publishes gathered data in a pre-defined format to the `Telemetry Response` topic. The format in which telemetry data is stored has to be defined by the backend operator beforehand. On reception of telemetry data packages by the backend, the data is parsed, stored in a database, and visualised on the backend's user interface. Figure 4.2 depicts the telemetry data procedure.

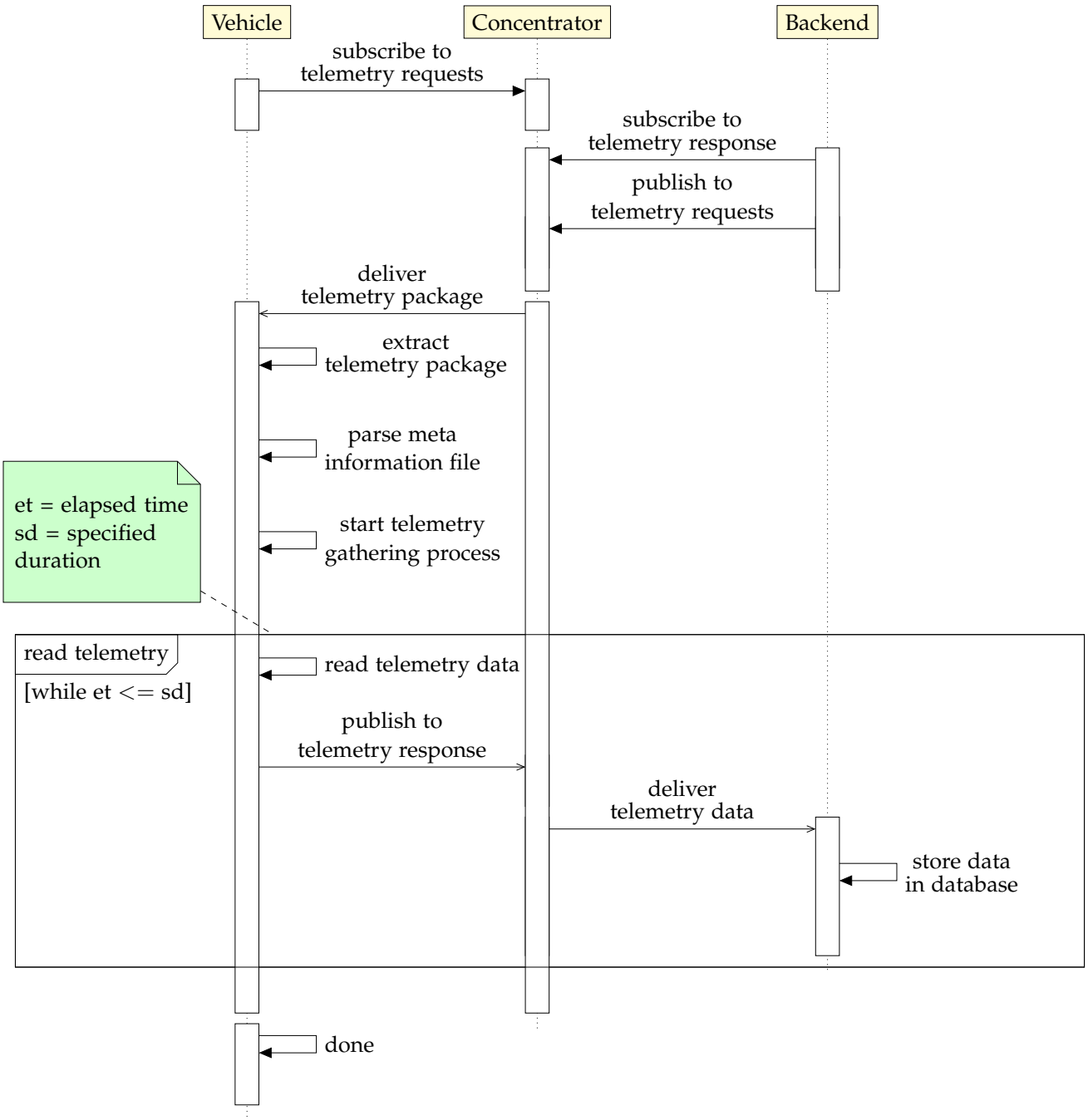

Figure 4.2: The telemetry data gathering sequence.

5 Implementation

In this chapter, we present the implementation of our concept. First, we focus on the framework. The framework is the foundation for secure communication between vehicles and infrastructure. We discuss used standards and technologies and show which role they take in our implementation. To demonstrate how our framework can be utilised, we present the implementation of the OTA and remote telemetry data gathering use cases. The ECU flashing and remote telemetry data gathering procedures are discussed in detail.

5.1 Framework

As described in Chapter 4, we use a meta-information file for storing use case-specific information. This meta-information file is then packed among use case-specific files into an archive. The archive is packed and encrypted according to the CMS standard and then deployed to the vehicle or infrastructure using the MQTT protocol. All tasks are managed and delegated to handlers by a single application, the controller. The software components of our framework are depicted in Figure 5.1 and discussed in detail hereafter.

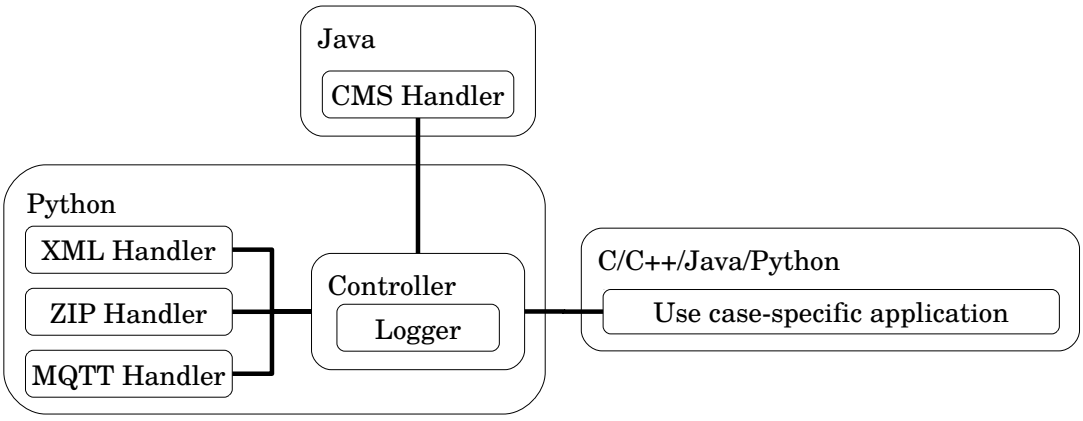

Figure 5.1: The framework's software components.

5.1.1 XML Handler

The format of the meta-information file follows the Extendable Markup Language (XML) format[1]. XML is a textual data format and often used for the definition of data structures. The fact that XML is a text-based format allows for high compression ratios when compressed by archiving applications such as `gzip`[2] or `7-Zip`[3]. This is especially useful for the deployment of archives where cell reception is bad, and network connections only have low bandwidth. An additional advantage of the XML file format is that it is widely supported. XML parsing tools and libraries are available for almost every programming language.

For the XML Handler we use the Python programming language. The Python standard library already contains an XML parsing module, namely `xml.etree.ElementTree`[4]. This module provides an easy-to-use application programming interface (API) for the creation and parsing of XML documents.

5.1.2 ZIP Handler

For packing multiple files into a single file, archiving or packing applications are used. These applications take multiple files as input, compress them, and pack them into a container file. A requirement for such applications is that files packed into an archive file are restorable without affecting their integrity. This property is called lossless. A widely used archive file format supporting the creation of lossless archive files is ZIP. Figure 5.2 depicts the packing and unpacking process of multiple files into a single ZIP container file.

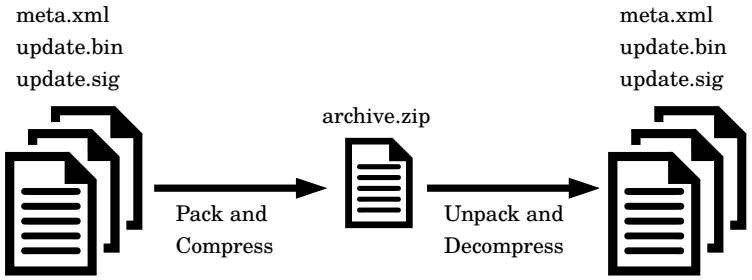

Figure 5.2: Packing/unpacking multiple files into/from a single ZIP container.

[1] `https://www.w3.org/TR/xml/`
[2] `https://www.gnu.org/software/gzip/`
[3] `https://www.7-zip.org/`
[4] `https://docs.python.org/3/library/xml.etree.elementtree.html`

Due to the widespread use of the ZIP file format, many tools and libraries for the handling thereof are available. We also use Python for the implementation of the ZIP handler. Python ships with the `zipfile`[5] module and fully supports the creation and extraction of ZIP files.

5.1.3 CMS Handler

The creation and parsing of CMS packets was implemented in JAVA using the `BouncyCastle`[6] library. BouncyCastle is open-source and provides APIs for handling CMS packets and certificates. Whereas X.509 certificates store public keys, the PKCS 12 format [36] is used for storing private keys. Key derivation, parsing of X.509 public key certificates and PKCS 12 private key certificates, and encryption/decryption of CMS packets, as discussed in Section 4.2.1, is also handled by the CMS application.

For interacting with the CMS application, an intuitive command-line interface is provided. Table 5.1 lists the available command-line switches for interacting with the CMS application. Listing 5.1 and Listing 5.2 show the command for packing and unpacking a CMS packet.

Command-line switch	**Description**
`-e, --encrypt`	Use encryption mode
`-d, --decrypt`	Use decryption mode
`-i <arg>, --input <arg>`	Path to input file
`-o <arg>, --output <arg>`	Path to output file
`-c <arg>, --certs <arg>`	Path to recipients certificates folder (encryption mode only)
`-s <arg>, --sender <arg>`	Path to sender private key certificate file (encryption mode only)
`-r <arg>, --recipient <arg>`	Path to recipient private key certificate file (decryption mode only)

Table 5.1: Command-line switches for interacting with the CMS application.

```
java -jar cms.jar -e -i archive.zip -o archive.zip.enc -c .\cert -s
   MQTT_Admin.p12
```

Listing 5.1: Command for creating an encrypted CMS packet.

[5] `https://docs.python.org/2/library/zipfile.html`
[6] `https://www.bouncycastle.org`

```
java -jar cms.jar -d -i archive.zip.enc -o archive.zip -c .\cert -r
   MQTT_Client.p12
```

Listing 5.2: Command for decrypting and unpacking an encrypted CMS packet.

5.1.4 MQTT Handler

The CMS packet is transmitted to the recipient via the MQTT protocol. Many open-source, e.g., Mosquitto[7], wolfMQTT[8], and commercial, e.g., HiveMQ[9],MQTTRoute[10], libraries supporting the MQTT protocol, are available. However, for our implementation, we used the Eclipse Paho MQTT libraries[11]. Eclipse Paho is open-source and available for many platforms and programming languages such as Java, Go, and Python. Our MQTT handler was implemented in Python.

The Eclipse Paho libraries abstract the low-level implementation details of the MQTT protocol and provide a clean interface for subscribing and publishing to MQTT topics. In our implementation, the first step was to create an MQTT client object. Our concept developed in Chapter 4 requires that all MQTT traffic should be encrypted and clients should be authenticated using TLS client authentication. Therefore, the TLS security parameters, including client certificate and supported cipher suite, were set accordingly using the tls_set method on the client object. The library also allows for specifying callback functions that are executed when a message was successfully published or received. Callback functions were defined by setting the on_publish and on_message properties of the previously instantiated client object.

After security parameters and publish/subscribe callback functions were set connections to the MQTT broker can be established. Listing 5.3 shows the code for creating a client object, setting the required TLS parameters, connecting to the MQTT message broker, and publishing a message to the MERCEDES/WDB9061551N47734/updatelog topic.

[7] https://mosquitto.org/
[8] https://www.wolfssl.com/products/wolfmqtt/
[9] https://www.hivemq.com/t/
[10] https://www.bevywise.com/mqtt-broker/
[11] https://www.eclipse.org/paho

```
import paho.mqtt.client as mqtt

def on_message_callback():
       pass

def on_publish_callback():
       pass

client = mqtt.Client()
client.tls_set(ca_certs = './AVL.crt', certfile = './MQTT_Client.p12',
   keyfile = './MQTT_Client.p12', ciphers = 'ECDHE-ECDSA-AES256-GCM-
   SHA384')
client.on_publish = on_publish_callback
client.on_message = on_message_callback
client.connect('127.0.0.1', '8885')
client.publish('MERCEDES/WDB9061551N47734/updatelog', payload = 'This is
    an example message.', qos = 2)
client.loop_forever()
```

Listing 5.3: Steps required to securely connect to the MQTT broker, running on 127.0.0.1:8885, and publish a message.

5.1.5 Controller

To combine the above-mentioned software components into a single application, a controller was implemented. The controller was written in Python and connects all handlers. The controller is responsible for delegating tasks to the appropriate handler and ensures proper error handling. The use case-specific application is also invoked and monitored by the controller. All errors and executed tasks are logged by a logging module and transmitted to the backend.

5.2 OTA Update

5.2.1 ECU Flashing Application

For this thesis, we focus on the ECU flashing procedure of a Hybrid Vehicle Control Unit (HVCU) that is used in a vehicle of the Chinese manufacturer JAC. ECU flashing is vendor depended, and the procedure is usually kept secret by the ECU manufacturers. However, for this thesis, we had access to the flashing procedure required to flash this specific ECU. This ECU requires to be flashed via the UDS protocol which further depends on the ISO-TP communication protocol.

We implemented the ECU flashing tool for the Linux operating system. The Linux kernel provides a kernel module for the ISO-TP protocol. This module abstracts low-level package handling according to the ISO-TP standard. Atop of the ISO-TP kernel module, we implemented the UDS protocol specific commands required for flashing the ECU. Accessing the kernel module-provided APIs requires a systems programming language. The programming language of choice for the ECU flashing application was, therefore, C++. Figure 5.3 depicts the architecture of the ECU flashing application.

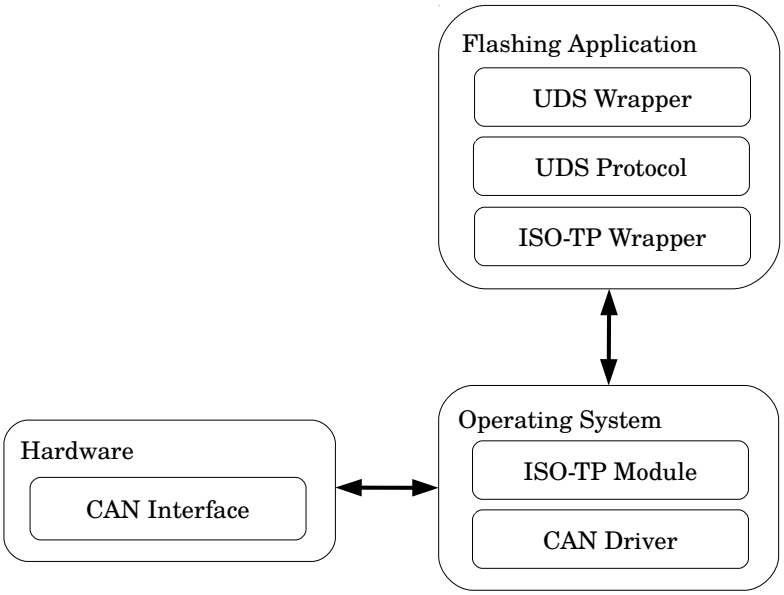

Figure 5.3: Architecture of the ECU flashing application.

UDS Flashing Procedure

The first step when flashing the ECU is to start an `Extended Diagnostic Session`. The `Extended Diagnostic Session` allows for disabling the non-diagnostic communication. Then, the ECU is instructed to enter a `Programming Session`. This is neccessary as rewriting the ECU's software is only possible when a `Programming Session` is active. However, to enter the high-privileged programming mode, the ECU flashing application is required to authorise itself via a challenge-response authorisation protocol.

The authorisation challenge is requested from the ECU by issuing a `Request Seed` command. The ECU responds with a challenge value. This challenge value is used by the ECU flashing application to calculate the correct response value. The calculated response value is then sent to ECU via the `Transfer Key` command. The challenge-response algorithm is vendor dependent and confidential. For this thesis, we had access to the ECU's challenge-response algorithm.

To write data to the ECU, the ECU flashing application first issues the `Write Data By Identifier` command with the ID of the flashing application as parameter. This command is used for writing the ID to the ECUs internal memory and is required for documentation purposes.

Before the ECU can be programmed, the ECU's memory has to be erased. For this purpose, an ECU internal function, responsible for erasing the internal memory, gets executed via the `Routine Control` command. When this routine is completed, the ECU needs to be prepared for the software download. This is done by issuing the `Request Download` command. After this step is completed, the ECU flashing application transmits the software via the `Transfer Data` command. When all data is transferred to the ECU, the `Request Transfer Exit` command is issued, signaling the ECU that the flashing application is done transmitting data.

To check the integrity of written data blocks an ECU internal function, responsible for calculating a Cyclic Redundancy Check (CRC) over the written blocks, is called, also via the `Routine Control` command. The CRC algorithm [41] is a widely used method for calculating checksums and detecting accidental data changes. Finally, the `ECU Reset` command is issued. This resets the ECU and the new software is loaded on startup. Figure 5.4 depicts the flashing procedure.

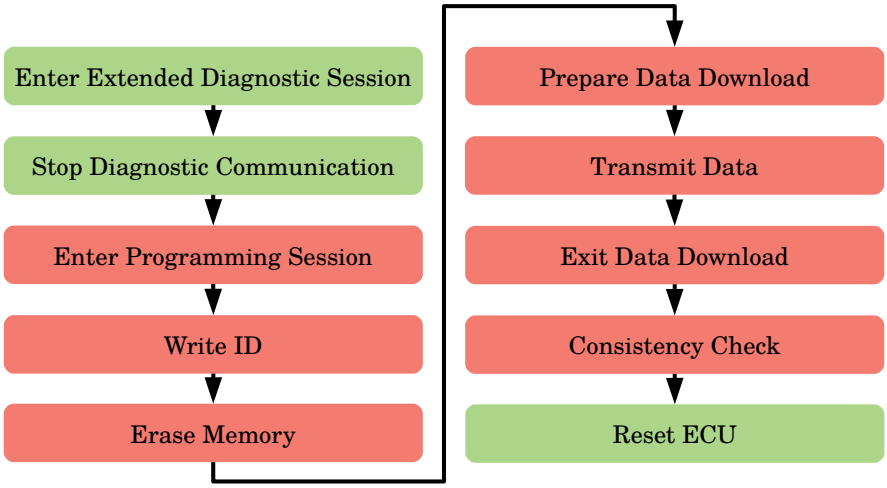

Figure 5.4: ECU flashing procedure. The blocks highlighted in green do not require authorisation; the blocks highlighted in red require authorisation.

5.2.2 MQTT Topic Structure

For update request, update package deployment and update log file reporting we use the skeleton shown in Listing 5.4. The `VMID` is a placeholder for the name of the vehicle manufacturer, e.g., MERCEDES, BMW, VW, whereas the `VIN` is replaced by the VIN of the vehicle that is requesting update information. The `mode` placeholder can be `updaterequests`, `updateresponse` or `updatelog`.

```
<VMID>/<VIN>/<mode>
```

Listing 5.4: MQTT Topic Structure Skeleton.

The following listings show examples of MQTT topics for the different modes.

```
MERCEDES/WDB9061551N47734/updaterequests
```

Listing 5.5: Update Request Topic Structure Example.

```
MERCEDES/WDB9061551N47734/updateresponse
```

Listing 5.6: Update Response Topic Structure.

```
MERCEDES/WDB9061551N47734/updatelog
```

Listing 5.7: Update Report Topic Structure.

5.2.3 Package Structure

The compressed update zip file contains all necessary information to validate and flash update files to corresponding ECUs. An example directory tree of an ECU update package is shown below.

```
update.zip
├── update0.s19
├── update0.sig
├── update1.s19
├── update1.sig
└── meta.xml
```

The update*.s19 files follow the Intel HEX file format specification [25] and are parsed, interpreted and flashed by the ECU flashing tool. The Intel HEX format specifies how binary information can be encoded into ASCII characters and stored in simple text files. Besides the update*.s19 files, update*.sig files are provided. These files store a signature calculated over the respective update file and are used for verifying the origin and correctness of update files. This provides an additional layer of defense.

The meta.xml file stores information about all update and signature files. By reading and interpreting the meta.xml file the flashing tool can match update files with the corresponding signature files.

5.2.4 XML File Structures

The XML structure for an update request is shown in Listing 5.8. The `timestamp` node stores a UNIX timestamp that is parsed by the backend and sent back to the vehicle in an update response messages. This timestamp is used for sanity checks. The `device` nodes give information on the ECUs installed in the vehicle. To identify the ECU, the `device id` attribute is provided. The element `version` contains the current firmware version of the ECU.

```
<update_request>
   <timestamp>UNIXTIMESTAMP</timestamp>
   <devices>
      <device id="...">version</device>
      <device id="...">version</device>
   </devices>
</update_request>
```

Listing 5.8: Skeleton of an XML file for an update request message.

Depending on whether updates are available, the backend packs different meta-information files into the update response archive. We discuss the two possible cases, 1) Updates available or 2) No updates available, hereafter.

Update(s) available

Listing 5.9 gives an example of how an update response file looks like in the case that updates are available. The `request_timestamp` node contains the timestamp from a previously sent update request. The update process allows for deploying updates for multiple devices with a single update package. For each update file, a `file` node is present. An update file entry is comprised of multiple elements specifying the filename, the file format, its size, the device id this update file is valid for and a signature file. Additionally, the `file order` attribute allows defining the update installation order. This is useful when updates depend on each other.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<update_response>
  <request_timestamp>UNIXTIMESTAMP</request_timestamp>
  <files>
    <file order="0">
      <name>update0.s19</name>
      <format>s19</format>
      <size unit="KB">100</size>
      <signature file="update0.sig" />
      <device>device-ID</device>
    </file>
    <file order="1">
      <name>update1.s19</name>
      <format>s19</format>
      <size unit="KB">1000</size>
      <signature file="update1.sig" />
      <device>device-ID</device>
    </file>
  </files>
</update_response>
```

Listing 5.9: Example of an XML file from an update response message in the case that updates are available.

No update(s) available

In the case that no updates are available, the XML file does not contain any `file` nodes. The controller interprets such an XML file as 'No updates available'. Listing 5.10 shows a response when no updates are available.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<update_response>
  <request_timestamp>UNIXTIMESTAMP</request_timestamp>
</update_response>
```

Listing 5.10: Example of an XML file from an update response message in the case that no updates are available.

5.3 Remote Telemetry Data Gathering

5.3.1 Telemetry Data Gathering Application

Opposed to ECU flashing, which requires the ECU flashing tool and the ECU to interact with each other, telemetry data gathering is a completely passive process. Every message that is sent on the CAN bus can be read by all CAN nodes without other nodes noticing. In these CAN messages various signals such as current engine rpm or vehicle velocity are encoded. This encoding is kept secret by car manufacturers. For this thesis, we had access to the CAN message signal encoding of the ECU that was also used for ECU flashing, an HVCU from the Chinese manufacturer JAC.

The telemetry data gathering application needs access to low-level system functions and hardware interfaces. Therefore, the application was implemented in C++, also for the Linux operating system. Figure 5.5 depicts the application's architecture.

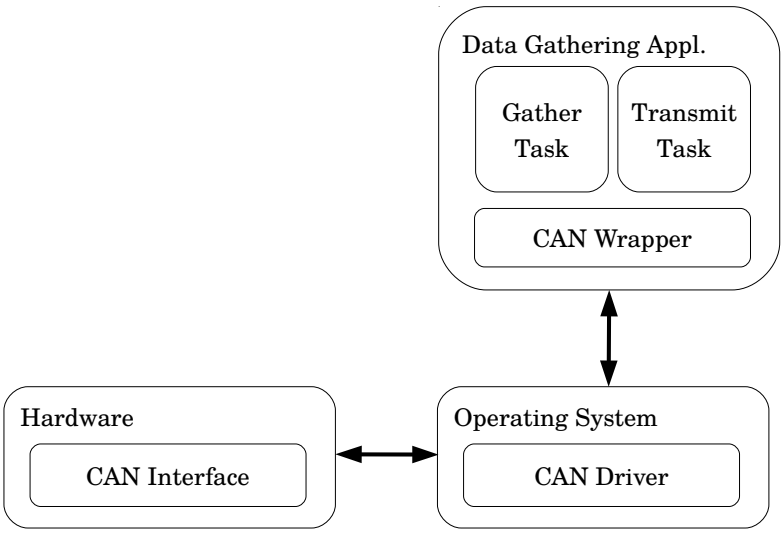

Figure 5.5: Architecture of the telemetry data gathering application.

Telemetry Data Gathering Procedure

The .dbc file, describing the CAN message layout, is parsed by the telemetry data gathering application. If the file is valid, the application initialises the CAN socket and applies CAN message filters such that only messages defined in the .dbc file are received. Then, two separate tasks are started. The first task reads messages from the CAN bus and extracts the values from the signals specified in the .dbc file. These values are stored in a buffering container. The second task runs periodically and takes the values from the buffering container, formats them in such a way that they can be interpreted by the backend, and passes the data to the framework. The framework then transmits the data to the backend. Telemetry data gathering is stopped when a pre-defined time, specified in the meta-information, elapsed.

5.3.2 MQTT Topic Structure

The MQTT topic skeleton for telemetry is based on the ECU update topic skeleton, shown in Listing 5.4. The difference, however, are the modes available. For the telemetry data gathering procedure we define two different topics, the `telemetryrequest` topic and the `telemetryresponse` topic. As soon as a vehicle is started, the vehicle component subscribes to the `telemetryrequest` topic and awaits a message from the backend. The backend deploys a telemetry package to the vehicle as soon as the backend operator wants to gather telemetry data. While the telemetry data gathering process is running, collected data is continuously published to the backend at

pre-defined intervals. In our implementation, we found an interval of one second suitable. This allows for almost real-time data gathering. The topic for publishing gathered telemetry data is `telemetryresponse`.

The following listings show MQTT topics for telemetry data gathering request and response.

```
MERCEDES/WDB9061551N47734/telemetryrequest
```

Listing 5.11: Telemetry Request Topic Structure Example.

```
MERCEDES/WDB9061551N47734/telemetryresponse
```

Listing 5.12: Telemetry Response Topic Structure.

5.3.3 Package Structure

Similar to the update process, a telemetry package is comprised of multiple files. The content of a telemetry archive looks like the following:

```
telemetry.zip
├── vehspeed.dbc
├── enginerpm.dbc
├── ...
├── combined.dbc
└── meta.xml
```

The *.dbc files contain all information necessary to extract signals from the CAN bus the backend operator is interested in. For example, the operator may be interested in the vehicle's speed, the engine RPM, or both. The DBC file format is a text-based format. Thus, high compression ratios are achieved when creating an ECU telemetry archive. An example of a DBC file used to extract vehicle speed, the status of the handbrake (pulled/released) and the position of the acceleration pedal (position in %) is shown in Listing 5.13.

```
BO_ 269 HVCU_FrP10: 8 HVCU
 SG_ HVCUVehSpd : 11|16@0+ (0.05625,0) [0|299.98] "km/h" EMS
 SG_ HVCUHndbrk : 41|1@0+ (1,0) [0|1] "" EMS

BO_ 267 HVCU_FrP09: 8 HVCU
 SG_ HVCUAccPedlPosn : 47|8@0+ (0.5,0) [0|100] "%" EMS
```

Listing 5.13: Example of a DBC file for reading multiple signals.

5.3.4 XML File Structure

The meta-information file stores information about a specific telemetry data gathering task. An example of a meta-information file is shown in Listing 5.14.

```
<?xml version="1.0" encoding="UTF-8"?>
  <telemetry_data>
    <dbc>vehspeed.dbc</dbc>
    <frequency>1000</frequency>
    <duration>60000</duration>
  </telemetry_data>
```

Listing 5.14: Example of a meta-information file for reading signals for 60 seconds with a sampling rate of 1 second.

The `dbc` element specifies the file where the signals the backend operator is interested in are defined. The `frequency` defines the sampling rate in milliseconds. The `duration` element specifies the time how long telemetry data should be gathered, also defined in milliseconds.

5.3.5 Telemetry Data File Format

Gathered telemetry data is stored in a file following the Comma Separated Value (CSV) format [48]. The CSV format is text-based and mostly used for data that can be stored in tabular form. A row in a CSV file corresponds to one data record. A data record has at least one field. Each field in a row is separated by a comma. The text-based format of CSV files allows for high compression ratios.

The first entry in a telemetry data CSV file is the header. The header consists of the two fixed strings ”Date” and ”Time”, and variable fields that store the name of each captured signal. One telemetry data entry consists of the current date, the time the message was recorded, and the extracted values.

Listing 5.15 shows data gathered in one of our tests. In this example the capture frequency was set to 10 milliseconds and two signals where captured, namely `HVCU_Cnt109` and `HVCU_Cnt10B`. These two signals count from 0 to 15. The counter is incremented every 10 milliseconds and restarts at 0 when the counter reaches 15.

```
Date;Time;HVCU_Cnt109;HVCU_Cnt10B
2017-07-27;13:37:49.260;14;14
2017-07-27;13:37:49.270;15;15
2017-07-27;13:37:49.280;0;0
2017-07-27;13:37:49.291;1;1
2017-07-27;13:37:49.301;2;2
2017-07-27;13:37:49.312;3;3
2017-07-27;13:37:49.322;4;4
2017-07-27;13:37:49.332;5;5
2017-07-27;13:37:49.343;6;6
2017-07-27;13:37:49.353;7;7
2017-07-27;13:37:49.363;8;8
2017-07-27;13:37:49.374;9;9
2017-07-27;13:37:49.384;10;10
2017-07-27;13:37:49.395;11;11
2017-07-27;13:37:49.405;12;12
2017-07-27;13:37:49.416;13;13
2017-07-27;13:37:49.427;14;14
2017-07-27;13:37:49.437;15;15
2017-07-27;13:37:49.447;0;0
```

Listing 5.15: Example of a gathered telemetry data stored in the CSV format.

6 Evaluation and Results

In this Chapter, we discuss the security review and penetration test which was conducted by an external company. We then describe how we set-up functionality tests and validated correct behavior. Finally, we discuss results of both security audit and functional tests.

6.1 Hardware Setup

To verify the functionality of our software components and the correctness of ECU flashing and telemetry data gathering, we installed our software on a custom built hardware designed by AVL. The board features an ARM-Cortex A9 connected to 2GB of RAM, an Ethernet interface, one serial interface and two CAN interfaces. We used Linux with a custom built kernel as the operating system. To reduce the attack surface, we stripped the self-compiled kernel from unnecessary features. A self-compiled kernel also allows for enabling advanced security features such as SELinux[1] and additional kernel modules, e.g., ISO-TP. The hardware setup is depicted in Figure 6.1.

We connected the AVL device via the LAN interface to a router. Internet connection is provided via an LTE dongle that is plugged-in to the router and allows for communication with the backend. As a testing ECU we used a Hybrid Vehicle Control Unit (HVCU) from a JAC S5 vehicle.

6.2 Security Audit and Penetration Test

Before the first information between the vehicle and backend was exchanged, we were required to check the setup for security vulnerabilities and misconfigurations. Therefore, a security review and penetration test was conducted by an external company.

[1] `https://selinuxproject.org/page/Main_Page`

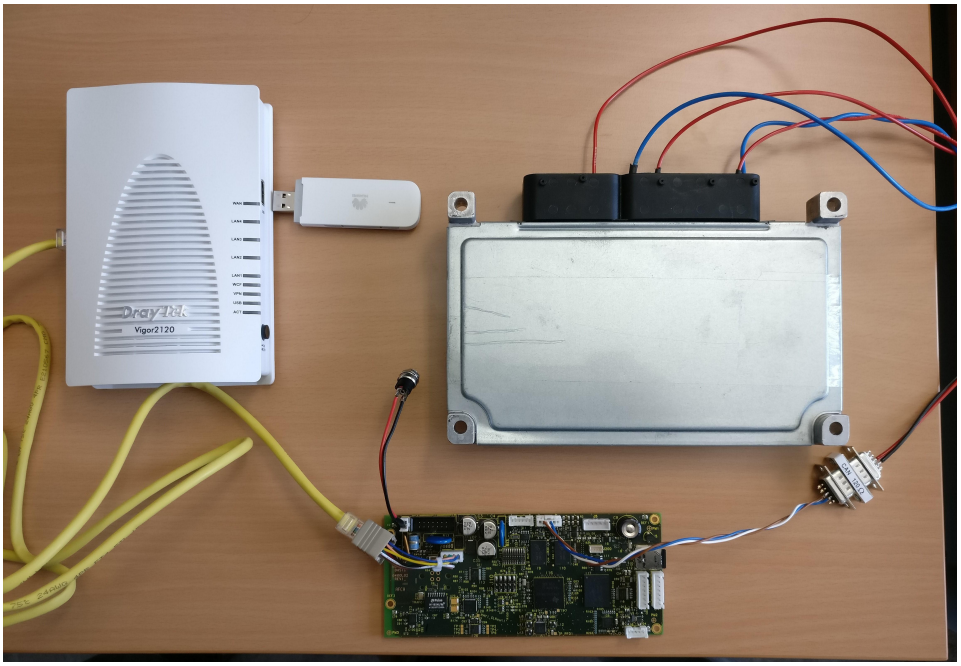

Figure 6.1: The hardware setup.

In principle, there are two types of penetration tests: white-box and black-box tests. Grey-box tests are also found in literature and are sometimes counted as a third penetration testing methodology. When a white-box test is conducted, the tester is provided with information about the system under test. Details about the inner workings of a system, e.g., used encryption algorithms or source code, are available to the tester. In contrast, when conducting a black-box test, no information is provided. A grey-box test is sort if in-between white- and black-box tests where some information is given.

However, the goal from a penetration tester's perspective is to extract confidential information or to gain unauthorised access to a system. In order to guarantee the highest level of security for our prototype, both white-box and black-box tests were conducted. The security review, as well as penetration tests, were carried out by an external company, specialised in security audits and penetration testing.

Besides the security features applied to our software, which we will discuss later, additional security measures were taken to harden the operating system running on the AVL device. We enabled the Linux kernel packet filtering module `netfilter`[2] and tuned its settings to match our security requirements. That is all ports, except the port used by OpenSSH[3] for device management purposes, were closed. In other words,

[2] `https://www.netfilter.org/`
[3] `https://www.openssh.com/`

all incoming connection requests to ports differing from the OpenSSH port were dropped. Additionally, all outgoing connections, except MQTT connections, were disallowed. In order to harden OpenSSH authentication, all authentication mechanisms, except public-key authentication[4], were turned off. As an additional layer of defense, only clients with certain IP addresses are allowed to connect to the OpenSSH server. Furthermore, all applications subject to low-level exploitation, e.g., the ECU flashing tool, were compiled with state-of-the-art exploit mitigation techniques such as position-independent executable (PIE), Non-executable stack (NX), stack canaries, or full Relocation Read-Only (RELRO). After applying the aforementioned security measures, the AVL device was ready for penetration testing.

The first test conducted was a black-box test. Access to the internal network was provided to the tester but no further information, e.g., IP addresses or open ports, was given. This simulates an attacker with physical access to a vehicle with a built-in AVL device. Since the penetration tester did not have any information about the device under attack, its first step was to enumerate devices on the network and look for the AVL device. Once the AVL device was found, it was scanned for open ports. This was followed by the tester trying to identify services running on the AVL device. The only service responding to identification requests was recognised as SSH service. The tester then attempted to establish an SSH connection to the AVL device. However, since access is restricted to specific IP addresses this attempt was not successful.

A real-world attacker would try to access the device via SSH even though the IP addresses SSH connections are restricted to are not known. Therefore, an attacker would assign all possible IP addresses to himself and check if connection requests are accepted. In order to simulate this attack, we removed the IP address restriction. This allowed the penetration tester to further investigate the OpenSSH service running on the AVL device. Due to public-key authentication, it was not possible for the simulated attacker to gain access to the AVL device. In the case that password-based authentication would have been enabled, the penetration tester would have tried to brute-force the password with a prepared list of most common passwords.

The tester's next step was trying to trick the router and AVL device into routing the traffic through the tester's device, i.e., Man-in-the-Middle. This attack allows for, if successful, eavesdropping and tampering on data. Such an attack is usually made by broadcasting forged ARP packets. This technique, known as ARP spoofing or ARP poisoning, adds invalid ARP entries to devices' ARP caches. The ARP spoofing attack was successful, and traffic was routed through the tester's device. However, since traffic between AVL device and concentrator is encrypted, it was not possible to extract confidential information. Moreover, as all traffic is authenticated, attempts to trick the concentrator into accepting and relaying unauthenticated messages, failed. The outcome of the attempted Man-in-the-Middle attack was that no information

[4] `https://kb.iu.edu/d/aews`

leakage or unauthenticated modification of data is possible.

After the black-box test was completed, the tester conducted a security review. Together with the penetration tester we reviewed our security concept and looked for potential vulnerabilities. We verified that used algorithms were safe to use and no flaws are publicly known. To further reduce the attack surface, router settings where reviewed, tweaked and set to the most defensive settings. We ensured that the firmware running on the router was up-to-date, disabled all unused services, hardened the router's authentication features, and secured administration interfaces.

To conclude, no severe vulnerabilities with regards to the assets and threats defined in Section 3 were discovered. Thus, we were ready to transmit production data and verify correct functionality of ECU flashing and remote telemetry data gathering.

6.3 Functionality Tests and Validation of Correctness

Conducting initial tests on an ECU already assembled into a vehicle is not encouraged due to difficulties that could arise when an ECU flashing procedure fails. A failed flashing attempt may cause a malfunctioning ECU and, thereby, an inoperable vehicle. Therefore, first tests were conducted in a lab environment.

To verify correct behavior of the ECU flashing and remote telemetry data gathering processes, we inspected how the ECU behaves when used with manufacturer-provided tools. We connected a CAN reader to the ECU's CAN interface and recorded all CAN messages the ECU transmits. These messages acted as a verification data set.

Both OTA Update and remote telemetry data gathering tests were first conducted in a lab environment. After we confirmed correct functionality a setup for in-vehicle tests was prepared. For comfortable transportation, we mounted the AVL Device and Router alongside a motorcycle battery on a panel. Figure 6.2 depicts the in-vehicle setup. All in-vehicle tests were conducted on AVL's private test track.

6.3.1 ECU Flashing

The first step was to verify correct behavior of the ECU flashing toolchain. During initial flashing experiments we encountered that the ECU does not send messages if the firmware update failed. A failed flashing attempt can have different reasons, e.g., writing incorrect data to the ECU's memory, errors during data transmission, or power failures. From there, we concluded that messages transmitted in precise intervals serve as a primary indicator for a successful flashing attempt.

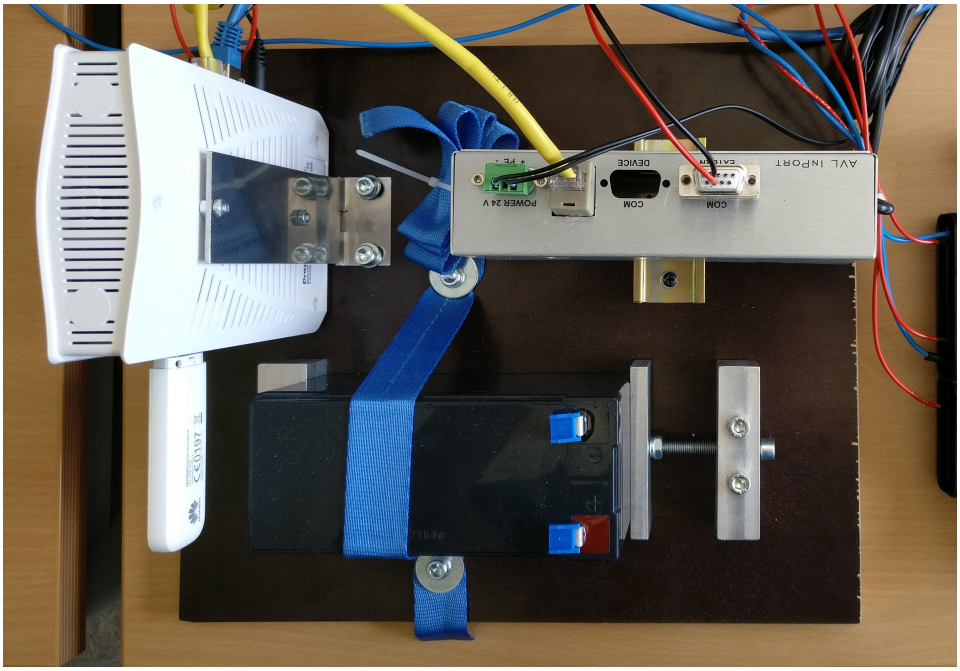

Figure 6.2: Hardware setup for in-vehicle tests.

With a success indicator defined, we were able to execute ECU flashing tests. Due to the complexity of the ECU flashing process in combination with OTA update package delivery these tests were carried out over the timespan of two months. As soon as teething troubles were eradicated and flashing attempts got stable, we verified the correctness of the ECU update toolchain in a real vehicle. We used a JAC S5 vehicle for our testing purposes.

As we left our lab environment, we had to look for new indicators of successful OTA update deployment. An evident and easy-to-test indicator is to check if it is possible to start the vehicle's engine after flashing the ECU. To take that a step further, a confident indicator is if the vehicle can be driven around without encountering failures. Additionally, since the test vehicle was also used for verification of functional safety systems, it was equipped with various measurement instruments and diagnostics devices that allowed for a more in-depth validation.

6.3.2 Telemetry Data Gathering

Similar to the ECU flashing process we first had to look for indicators confirming successful telemetry data gathering attempts. Initial tests where conducted in a lab environment. The data transmitted in messages in a lab setting, however, is mostly meaningless. In the lab setting it is not possible to validate the vehicle's speed, the

engine rpm, or current gear. The ECU, however, transmits signals acting as a counter on almost all CAN IDs. These counters count from 0 to 15 and are reset to 0 when the counter reaches the value 15. Our validation of telemetry data gathering in the lab environment relied on these counters. Compared to the ECU flashing, reading telemetry is a purely passive process and, therefore, it is almost impossible to cause a vehicle to malfunction. Retrieving data from the CAN bus and interpreting CAN bus data correctly is much lower in complexity and not as a critical process as updating an ECU's firmware. However, as discussed in previous sections, software bugs or failing hardware could cause invalid messages being written to the CAN bus.

To validate our implementation against the defined indicators we first validated the ECU telemetry gathering application. For this purpose, DBC files containing various signals, including the aforementioned counters, where defined. For each CAN ID and message, an individual DBC file was prepared. To verify correct behavior of the ECU telemetry reader, we started the application with different interval parameters. Counter values were extracted every 10 ms and gradually increased every 10 ms. For every frequency step, all messages were recorded for a timespan of five minutes. This recording process was followed by manually verifying the counters encoded in the messages against data captured by a third-party CAN sniffing tool. As a final step, extracted data was sent to the backend and visualised for further inspection. The successful validation of remote telemetry data gathering in the lab environment allowed for testing in the JAC S5 vehicle.

We also used the JAC S5 vehicle for testing remote telemetry data gathering. The built-in measurement and diagnostic devices came in handy in verifying the data gathering procedure. In contrast to the tests conducted in a lab environment, where it was only possible to verify data by checking the counter signals, we had many more valid signals available. Therefore, new DBC files were specified. To validate our implementation, data was recorded continuously and sent to the backend while driving the JAC S5 around AVL's test track. This allowed for almost real-time verification of the recorded data. Data sent to the backend was verified in various ways. Easily verifiable signals such as current vehicle velocity or engine rpm were checked against data displayed in the tachometer. The current gear extracted from CAN messages was checked against the gear stick position in the vehicle. Other signals were verified by comparing them against signals captured by built-in diagnostics devices.

6.4 Results

6.4.1 Security Audit and Penetration Test

The results of the conducted black-box test show that it is not possible to gain unauthorised access to the AVL device. Moreover, the tester did not manage to exfiltrate confidential information by eavesdropping on network traffic. Additionally, all data tampering attempts failed. During the black-box test, only one minor security issue was found. The OpenSSH service running on the AVL device was outdated, and various vulnerabilities are publicly known, e.g., CVE-2016-8858[5] or CVE-2016-6515[6]. However, these vulnerabilities only allowed for Denial of Service attacks. These attacks can easily be mitigated by upgrading OpenSSH to the latest available version.

The security audit confirmed that the security concept is sound and no severe publicly known vulnerabilities exist, neither in the utilised software nor the used protocols and cryptographic algorithms. However, two issues arose during the review:

1. Storage of cryptographic key material and secure cryptographic operations
2. Tamper-proof system's software.

We will address these issues and discuss possible solutions in Chapter 7.

6.4.2 Functionality: OTA update

After ECU updates proofed to be reliable in a lab environment, real tests were conducted in a JAC S5 vehicle. All ECU update attempts were successful, and the vehicle was fully operable at all times. Also, the diagnostics and measurement devices did not report any anomalies. The time taken to complete a full OTA update procedure, from OTA update initiation to update log file publishing, was roughly four minutes. Most of the time is spent during the actual ECU flashing process and depends on bus bandwidth, firmware update size, and the ECU's capabilities. For example, connecting and updating an ECU via FlexRay[7,8] (max. Bandwidth: 10 MBit/s) instead of CAN (max. Bandwidth: 1 MBit/s) could decrease the time needed for the flashing process by a factor of ten. However, the time required for a ECU flashing is almost constant and varies only in the range of seconds for identically constructed vehicles. The deployment of an OTA update package, however, may vary

[5] https://nvd.nist.gov/vuln/detail/CVE-2016-8858
[6] https://nvd.nist.gov/vuln/detail/CVE-2016-6515
[7] https://www.iso.org/standard/59809.html
[8] https://www.iso.org/standard/59804.html

and is heavily dependent on cellular network reception and bandwidth. In our case, the time needed from OTA update initiation to complete reception of the OTA update package was roughly 10 seconds. The AVL Device was connected via a 4G wireless mobile broadband stick. Table 6.1 shows the mean time needed for an OTA update.

Task	**Time taken [s]**
Update request encryption	6.4
Update request publishing	2.7
Update response reception	2.5
Update response decryption	6.9
ECU flashing	218.7
Update log encryption	7.2
Update log publishing	2.7
Overall	247.1

Table 6.1: Time taken for a complete OTA update procedure.

It is noteworthy that we did not take any safety measures into account. Before deploying an OTA update system, it is crucial to consider safety. For example, the AVL Device must not initiate ECU flashing while the vehicle is being operated. In addition, one should also think about designing ECUs that are resilient to failed ECU flashing attempts. These issues are further discussed in Section 7.2

6.4.3 Functionality: Remote Telemetry Data Gathering

The data gathered by the AVL Device was sent to the backend and verified against the dashboard and the values reported by diagnostic and measurement devices. During our tests, no discrepancy between actual and reported values was encountered. Figure 6.3 illustrates the vehicle's velocity over a timespan of 20 seconds as visualised in the backend.

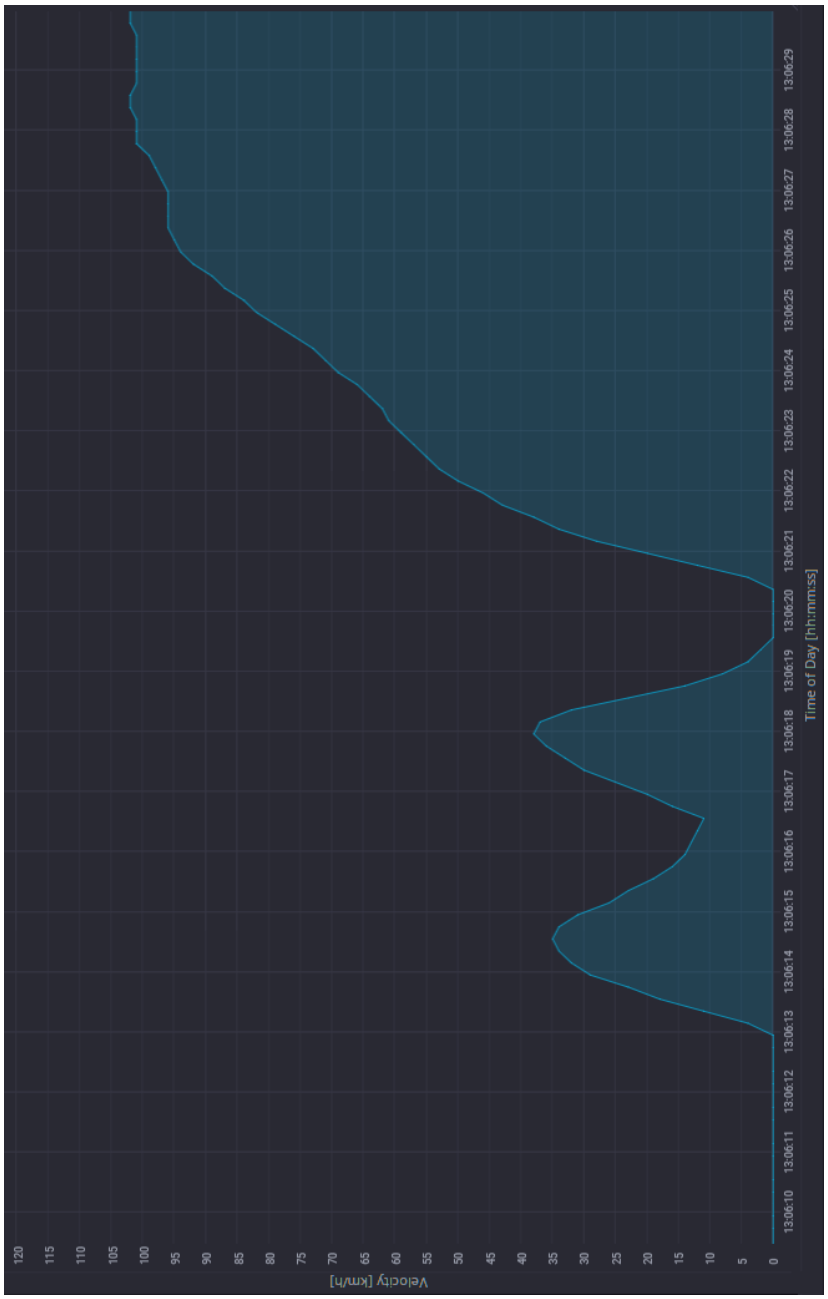

Figure 6.3: Visualisation of the vehicle's velocity [km/h] recorded over a timespan of 20 seconds.

7 Conclusion and Future Work

7.1 Conclusion

In this work, we analysed state-of-the-art vehicles for potential cyber security threats. In particular, we looked at recent attacks on vehicles and the impact thereof with regards to vehicle safety and security. Based on the analysis of previous attacks we designed a basic framework for secure communication between vehicles and a backend. To secure the framework and extract potential security pitfalls, we applied the STRIDE threat modeling methodology. Therefore, we defined assets, elaborated on different types of attackers, and specified a comprehensive attacker model. Based on the findings of the threat modeling process, we incorporated state-of-the-art security measures. Our security concept uses modern public-key and symmetric cryptographic primitives and heavily relies on the security thereof. For an in-depth defense, we proposed a three-layer model, with different security objectives for each layer.

To validate our secure communication framework, we designed a methodology for deploying OTA updates to vehicles. Furthermore, as a second use-case, we proposed a design for remotely gathering telemetry data and transmitting data at regular intervals to a backend. Both OTA update and remote telemetry data gathering use-cases were implemented and verified.

We evaluated our concept in two ways. First, the concept was reviewed and stress-tested for security vulnerabilities by an external company specialised in cyber security. Second, correct functionality of the OTA and remote telemetry data gathering was verified. Therefor an ECU OTA update was deployed and installed on a vehicle. Moreover, real-world telemetry data was gathered while the vehicle was operated and visualised in the backend.

In conclusion, with modern vehicles becoming increasingly connected, it is clear that cyber security plays an integral role. Due to the complexity of today's software, it is crucial to fix bugs as early as possible. However, as we have seen, developing a framework for secure communication is not a trivial task and prone to errors.

7.2 Future Work

The focus of this thesis was to develop a framework for secure communication between vehicles and infrastructure. Novel vehicles, however, do not only communicate with infrastructure, but also incorporate so-called vehicle-to-vehicle communication. From a security perspective, such systems must provide a high level of security. Attacking vehicle-to-vehicle communication by, for example, communicating wrong information to nearby vehicles may result in fatal crashes.

Another important research topic is secure in-vehicular communication. Currently, bus systems are accessible to all components connected to the bus. This imposes severe security-related implications. For example, an attacker could implant a malicious component into the in-vehicular network and spoof other components or eavesdrop on the bus traffic.

An additional research topic may focus on the safety perspective of OTA updates. In our implementation, an ECU update is applied to the ECU as soon as the update is deployed. This may result in an inoperable vehicle or fatal crash when the vehicle is being operated. In addition, when designing and implementing OTA mechanisms, one should think about incorporating fallback mechanisms. An inoperable vehicle due to a failed flashing attempt would not only result in upset customers, but also in expensive repair costs.

To summarise, the ongoing fast-paced development of vehicles unveils new exciting research topics. The fields of connected cars and autonomous driving are increasingly important. Therefore, more research is required for enabling safe and secure vehicles of tomorrow.

Appendix

Threat Modeling Report

Threat Model Summary:

Not Started	0
Not Applicable	6
Needs Investigation	0
Mitigation Implemented	20
Total	26
Total Migrated	0

---

1. Spoofing of the Concentrator External Destination Entity [State: Mitigation Implemented] [Priority: High]

Category:	Spoofing
Description:	Concentrator may be spoofed by an attacker and this may lead to data being sent to the attacker's target instead of Concentrator. Consider using a standard authentication mechanism to identify the external entity.
Justification:	Incorporate authentication mechanisms

2. External Entity Concentrator Potentially Denies Receiving Data [State: Mitigation Implemented] [Priority: High]

Category:	Repudiation
Description:	Concentrator claims that it did not receive data from a process on the other side of the trust boundary. Consider using logging or auditing to record the source, time, and summary of the received data.
Justification:	Incorporate digital signatures and logging mechanisms

3. Data Flow receive Is Potentially Interrupted [State: Not Applicable] [Priority: Medium]

Category:	Denial Of Service
Description:	An external agent interrupts data flowing across a trust boundary in either direction.
Justification:	DoS is out of scope

4. Elevation by Changing the Execution Flow in Data Processor (Vehicle) [State: Mitigation Implemented] [Priority: High]

Category: Elevation Of Privilege

Description: An attacker may pass data into Data Processor (Vehicle) in order to change the flow of program execution within Data Processor (Vehicle) to the attacker's choosing.

Justification: Incorporate authenticated encryption and validate input

5. Data Processor (Vehicle) May be Subject to Elevation of Privilege Using Remote Code Execution [State: Mitigation Implemented] [Priority: High]

Category: Elevation Of Privilege

Description: Concentrator may be able to remotely execute code for Data Processor (Vehicle).

Justification: Incorporate authenticated encryption and validate input

6. Elevation Using Impersonation [State: Mitigation Implemented] [Priority: High]

Category: Elevation Of Privilege

Description: Data Processor (Vehicle) may be able to impersonate the context of Concentrator in order to gain additional privilege.

Justification: Incorporate authentication mechanisms

7. Data Flow receive Is Potentially Interrupted [State: Not Applicable] [Priority: Low]

Category: Denial Of Service

Description: An external agent interrupts data flowing across a trust boundary in either direction.

Justification: DoS is out of scope

8. Potential Process Crash or Stop for Data Processor (Vehicle) [State: Not Applicable] [Priority: Low]

Category: Denial Of Service

Description: Data Processor (Vehicle) crashes, halts, stops or runs slowly; in all cases violating an availability metric.

Justification: DoS is out of scope

9. Data Flow Sniffing [State: Mitigation Implemented] [Priority: High]

Category: Information Disclosure

Description: Data flowing across receive may be sniffed by an attacker. Depending on what type of data an attacker can read, it may be used to attack other parts of the system or simply be a disclosure of information leading to compliance violations. Consider encrypting the data flow.

Justification: Incorporate (authenticated) encryption

10. Potential Data Repudiation by Data Processor (Vehicle) [State: Mitigation Implemented] [Priority: High]

Category: Repudiation

Description: Data Processor (Vehicle) claims that it did not receive data from a source outside the trust boundary. Consider using logging or auditing to record the source, time, and summary of the received data.

Justification: Incorporate digital signatures and logging mechanisms

11. Potential Lack of Input Validation for Data Processor (Vehicle) [State: Mitigation Implemented] [Priority: High]

Category: Tampering

Description: Data flowing across receive may be tampered with by an attacker. This may lead to a denial of service attack against Data Processor (Vehicle) or an elevation of privilege attack against Data Processor (Vehicle) or an information disclosure by Data Processor (Vehicle). Failure to verify that input is as expected is a root cause of a very large number of exploitable issues. Consider all paths and the way they handle data. Verify that all input is verified for correctness using an approved list input validation approach.

Justification: Incorporate authenticated encryption and validate input

12. Spoofing the Concentrator External Entity [State: Mitigation Implemented] [Priority: High]

Category: Spoofing

Description: Concentrator may be spoofed by an attacker and this may lead to unauthorized access to Data Processor (Vehicle). Consider using a standard authentication mechanism to identify the external entity.

Justification: Incorporate authentication mechanisms

13. Spoofing the Data Processor (Vehicle) Process [State: Mitigation Implemented] [Priority: High]

Category: Spoofing

Description: Data Processor (Vehicle) may be spoofed by an attacker and this may lead to information disclosure by Concentrator. Consider using a standard authentication mechanism to identify the destination process.

Justification: Incorporate authentication mechanisms

14. Elevation by Changing the Execution Flow in Data Processor (Vehicle) [State: Mitigation Implemented] [Priority: High]

Category: Elevation Of Privilege

Description: An attacker may pass data into Data Processor (Vehicle) in order to change the flow of program execution within Data Processor (Vehicle) to the attacker's choosing.

Justification: Incorporate authenticated encryption and validate input

15. Data Processor (Vehicle) May be Subject to Elevation of Privilege Using Remote Code Execution [State: Mitigation Implemented] [Priority: High]

Category: Elevation Of Privilege

Description: Concentrator may be able to remotely execute code for Data Processor (Vehicle).

Justification: Incorporate authenticated encryption and validate input

16. Elevation Using Impersonation [State: Mitigation Implemented] [Priority: High]

Category: Elevation Of Privilege

Description: Data Processor (Vehicle) may be able to impersonate the context of Concentrator in order to gain additional privilege.

Justification: Incorporate authentication mechanisms

17. Data Flow transmit Is Potentially Interrupted [State: Not Applicable] [Priority: Low]

Category: Denial Of Service

Description: An external agent interrupts data flowing across a trust boundary in either direction.

Justification: DoS is out of scope

18. Potential Process Crash or Stop for Data Processor (Vehicle) [State: Not Applicable] [Priority: Medium]

Category: Denial Of Service

Description: Data Processor (Vehicle) crashes, halts, stops or runs slowly; in all cases violating an availability metric.

Justification: DoS is out of scope

19. Data Flow Sniffing [State: Mitigation Implemented] [Priority: High]

Category: Information Disclosure

Description: Data flowing across transmit may be sniffed by an attacker. Depending on what type of data an attacker can read, it may be used to attack other parts of the system or simply be a disclosure of information leading to compliance violations. Consider encrypting the data flow.

Justification: Incorporate (authenticated) encryption

20. Potential Data Repudiation by Data Processor (Vehicle) [State: Mitigation Implemented]

[Priority: High]

Category: Repudiation

Description: Data Processor (Vehicle) claims that it did not receive data from a source outside the trust boundary. Consider using logging or auditing to record the source, time, and summary of the received data.

Justification: Incorporate digital signatures and logging mechanisms

21. Potential Lack of Input Validation for Data Processor (Vehicle) [State: Mitigation Implemented] [Priority: High]

Category: Tampering

Description: Data flowing across transmit may be tampered with by an attacker. This may lead to a denial of service attack against Data Processor (Vehicle) or an elevation of privilege attack against Data Processor (Vehicle) or an information disclosure by Data Processor (Vehicle). Failure to verify that input is as expected is a root cause of a very large number of exploitable issues. Consider all paths and the way they handle data. Verify that all input is verified for correctness using an approved list input validation approach.

Justification: Incorporate authenticated encryption and validate input

22. Spoofing the Concentrator External Entity [State: Mitigation Implemented] [Priority: High]

Category: Spoofing

Description: Concentrator may be spoofed by an attacker and this may lead to unauthorized access to Data Processor (Vehicle). Consider using a standard authentication mechanism to identify the external entity.

Justification: Incorporate authentication mechanisms

23. Spoofing the Data Processor (Vehicle) Process [State: Mitigation Implemented] [Priority: High]

Category: Spoofing

Description: Data Processor (Vehicle) may be spoofed by an attacker and this may lead to information disclosure by Concentrator. Consider using a standard authentication mechanism to identify the destination process.

Justification: Incorporate authentication mechanisms

24. Data Flow transmit Is Potentially Interrupted [State: Not Applicable] [Priority: Low]

Category: Denial Of Service

Description: An external agent interrupts data flowing across a trust boundary in either direction.

Justification: DoS is out of scope

25. External Entity Concentrator Potentially Denies Receiving Data [State: Mitigation Implemented] [Priority: High]

Category:	Repudiation
Description:	Concentrator claims that it did not receive data from a process on the other side of the trust boundary. Consider using logging or auditing to record the source, time, and summary of the received data.
Justification:	Incorporate digital signatures and logging mechanisms

26. Spoofing of the Concentrator External Destination Entity [State: Mitigation Implemented] [Priority: High]

Category:	Spoofing
Description:	Concentrator may be spoofed by an attacker and this may lead to data being sent to the attacker's target instead of Concentrator. Consider using a standard authentication mechanism to identify the external entity.
Justification:	Incorporate authentication mechanisms

Bibliography

[1] Processing Standards Publications 197. *Specification for the Advanced Encryption Standard (AES)*. Standard. Gaithersburg, MD, United States: National Institute of Standards & Technology, 2001 (cit. on p. 14).

[2] Marwan Abi-Antoun, Daniel Wang, and Peter Torr. "Checking Threat Modeling Data Flow Diagrams for Implementation Conformance and Security." In: *Proceedings of the Twenty-second IEEE/ACM International Conference on Automated Software Engineering*. ASE '07. Atlanta, Georgia, USA: ACM, 2007, pp. 393–396. ISBN: 978-1-59593-882-4. DOI: 10.1145/1321631.1321692 (cit. on p. 29).

[3] Manos Antonakakis et al. "Understanding the Mirai Botnet." In: *26th USENIX Security Symposium (USENIX Security 17)*. Vancouver, BC: USENIX Association, 2017, pp. 1093–1110. ISBN: 978-1-931971-40-9 (cit. on p. 2).

[4] A. Banks and Gupta R. *MQTT Version 3.1.1*. Specification. OASIS, Dec. 2015 (cit. on p. 16).

[5] Elaine Barker, Don Johnson, and Miles Smid. *NIST Special Publication 800-56A: Recommendation for Pair-Wise Key Establishment Schemes Using Discrete Logarithm Cryptography (Revised)*. Standard. Gaithersburg, MD, United States: National Institute of Standards & Technology, 2007 (cit. on p. 37).

[6] William C. Barker and Elaine B. Barker. *SP 800-67 Rev. 1. Recommendation for the Triple Data Encryption Algorithm (TDEA) Block Cipher*. Standard. Gaithersburg, MD, United States: National Institute of Standards & Technology, 2012 (cit. on p. 14).

[7] Britt Blackwelder et al. *The Volkswagen Scandal*. Case Study. University of Richmond: Robins School of Business, 2016 (cit. on p. 1).

[8] Toni Celina. "Secure Software Remote Update and Diagnosis - Backend." MA thesis. Graz University of Technology, 2016 (cit. on p. 36).

[9] CERT-EU. *WannaCry Ransomware Campaign Exploiting SMB Vulnerability*. CERT-EU Security Advisory 2017-012. 2017 (cit. on p. 2).

[10] Stephen Checkoway et al. "Comprehensive Experimental Analyses of Automotive Attack Surfaces." In: *Proceedings of the 20th USENIX Conference on Security*. SEC'11. San Francisco, CA: USENIX Association, 2011, pp. 6–6. URL: http://dl.acm.org/citation.cfm?id=2028067.2028073 (cit. on p. 3).

[11] Claude Le Pape et al. *DELIVERABLE D1.3 of Work Package 1: Generation 1 demonstrations, conclusions, and perspectives.* 2014. URL: http://www.arrowhead.eu (visited on 09/01/2017) (cit. on pp. 11, 12).

[12] D. Cooper et al. *Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile*. RFC 5280. IETF, May 2008 (cit. on p. 37).

[13] Microsoft Corporation. *Improving Web Application Security: Threats and Countermeasures*. Microsoft Press, 2003. ISBN: 0735618429 (cit. on p. 28).

[14] Quynh H. Dang. *Secure Hash Standard*. Standard. Gaithersburg, MD, United States: National Institute of Standards & Technology, 2015 (cit. on p. 38).

[15] Solar Designer. *Non-executable User Stack*. 1997 (cit. on p. 7).

[16] T. Dierks. *The Transport Layer Security (TLS) Protocol Version 1.2*. RFC 5246. IETF, Aug. 2008 (cit. on p. 15).

[17] W. Diffie and M. Hellman. "New directions in cryptography." In: *IEEE Transactions on Information Theory* 22.6 (Aug. 1976), pp. 644–654. ISSN: 0018-9448. DOI: 10.1109/TIT.1976.1055638 (cit. on pp. 14, 37).

[18] Thai Duong and Juliano Rizzo. *Here Come The ⊕ Ninjas*. 2011. URL: https://nerdoholic.org/uploads/dergln/beast_part2/ssl_jun21.pdf (visited on 11/08/2017) (cit. on p. 15).

[19] Morris Dworkin. *NIST Special Publication 800-38D: Recommendation for Block Cipher Modes of Operation: Galois/Counter Mode (GCM) and GMAC*. Standard. Gaithersburg, MD, United States: National Institute of Standards & Technology, 2007 (cit. on p. 37).

[20] EVITA. *E-safety vehicle intrusion protected applications*. 2008. URL: https://www.evita-project.org/ (visited on 08/16/2017) (cit. on pp. 10, 11).

[21] N. J. Al Fardan and K. G. Paterson. "Lucky Thirteen: Breaking the TLS and DTLS Record Protocols." In: *2013 IEEE Symposium on Security and Privacy*. May 2013, pp. 526–540. DOI: 10.1109/SP.2013.42 (cit. on p. 15).

[22] R. Housley. *Cryptographic Message Syntax (CMS)*. RFC 5652. IETF, Sept. 2009 (cit. on p. 15).

[23] R. Housley. *Cryptographic Message Syntax (CMS) Authenticated-Enveloped-Data Content Type*. RFC 5083. IETF, Nov. 2007 (cit. on p. 37).

[24] Troy Hunt. *Controlling vehicle features of Nissan LEAFs across the globe via vulnerable APIs*. 2016. URL: https://www.troyhunt.com/controlling-vehicle-features-of-nissan/ (visited on 08/18/2017) (cit. on p. 6).

[25] Intel. *Hexadecimal Object File Format Specification*. Specification. Jan. 1988 (cit. on p. 52).

[26] *Road vehicles – Controller area network (CAN)*. Standard. Geneva, CH: International Organization for Standardization, Dec. 2015 (cit. on p. 18).

[27] *Unified diagnostic services (UDS) – Part 1: Specification and requirements*. Standard. Geneva, CH: International Organization for Standardization, Dec. 2013 (cit. on p. 20).

[28] *Road vehicles – Diagnostic communication over Controller Area Network (DoCAN) – Part 2: Transport protocol and network layer services*. Standard. Geneva, CH: International Organization for Standardization, Dec. 2016 (cit. on p. 19).

[29] Don Johnson, Alfred Menezes, and Scott Vanstone. "The elliptic curve digital signature algorithm (ECDSA)." In: *International Journal of Information Security* 1.1 (2001), pp. 36–63 (cit. on pp. 15, 38).

[30] Samy Kamkar. *OwnStar: Locates, Unlocks, Remote Starts GM/OnStar Cars*. 2015. URL: https://samy.pl/ (visited on 08/10/2017) (cit. on p. 3).

[31] Trishank Karthik et al. "Uptane: Securing Software Updates for Automobiles." In: *Embedded Security in Cars (ESCAR)* (2016) (cit. on pp. 7, 9, 39).

[32] Karl Koscher et al. "Experimental Security Analysis of a Modern Automobile." In: *31st IEEE Symposium on Security and Privacy, S&P 2010, 16-19 May 2010, Berleley/Oakland, California, USA*. 2010, pp. 447–462. DOI: 10.1109/SP.2010.34. URL: https://doi.org/10.1109/SP.2010.34 (cit. on p. 3).

[33] C. Lesjak et al. "Securing smart maintenance services: Hardware-security and TLS for MQTT." In: *2015 IEEE 13th International Conference on Industrial Informatics (INDIN)*. July 2015, pp. 1243–1250. DOI: 10.1109/INDIN.2015.7281913 (cit. on p. 37).

[34] Charlie Miller and Chris Valasek. *Remote exploitation of an unaltered passenger vehicle*. Tech. rep. Black Hat USA, 2015 (cit. on p. 3).

[35] Bodo Möller, Thai Duong, and Krzysztof Kotowicz. *This POODLE bites: exploiting the SSL 3.0 fallback*. 2014. URL: https://www.openssl.org/~bodo/ssl-poodle.pdf (visited on 11/08/2017) (cit. on p. 15).

[36] K. Moriarty et al. *PKCS #12: Personal Information Exchange Syntax v1.1*. RFC 7292. IETF, July 2014 (cit. on p. 46).

[37] J. Mössinger. "Software in Automotive Systems." In: *IEEE Software* 27.2 (Mar. 2010), pp. 92–94. ISSN: 0740-7459. DOI: 10.1109/MS.2010.55 (cit. on p. 1).

[38] Michael Muckin and Scott C Fitch. *A Threat-Driven Approach to Cyber Security*. Tech. rep. Lockheed Martin, 2014 (cit. on pp. 23, 24).

[39] Suvda Myagmar, Adam J. Lee, and William Yurcik. "Threat modeling as a basis for security requirements." In: *Proceedings of the IEEE Symposium on Requirements Engineering for Information Security*. 2005 (cit. on p. 28).

[40] Sen Nie, Ling Liu, and Yuefeng Du. *Free-Fall: Hacking Tesla from Wireless to CAN Bus*. Tech. rep. Black Hat USA, 2017 (cit. on p. 4).

[41] W. W. Peterson and D. T. Brown. "Cyclic Codes for Error Detection." In: *Proceedings of the IRE* 49.1 (Jan. 1961), pp. 228–235. ISSN: 0096-8390. DOI: 10.1109/JRPROC.1961.287814 (cit. on p. 50).

[42] J. Randall et al. *Use of the RSA-KEM Key Transport Algorithm in the Cryptographic Message Syntax (CMS)*. RFC 5990. IETF, Sept. 2010 (cit. on p. 37).

[43] Marco Frigessi di Rattalma. *The Dieselgate: A Legal Perspective*. Springer, 2017 (cit. on p. 1).

[44] E. Rescorla. *The Transport Layer Security (TLS) Protocol Version 1.3 draft-ietf-tls-tls13-11*. RFC. IETF, Dec. 2015 (cit. on p. 15).

[45] E. Rescorla. *TLS Elliptic Curve Cipher Suites with SHA-256/384 and AES Galois Counter Mode (GCM)*. RFC 5289. IETF, Aug. 2008 (cit. on p. 39).

[46] Robert Lemos. *Script kiddies: The Net's cybergangs*. 2000. URL: http://www.zdnet.com/article/script-kiddies-the-nets-cybergangs/ (visited on 09/18/2017) (cit. on p. 25).

[47] Sebastian Anthony. *The first rule of zero-days is no one talks about zero-days (so we'll explain)*. 2015. URL: https://arstechnica.com/information-technology/2015/10/the-rise-of-the-zero-day-market/ (visited on 09/18/2017) (cit. on p. 26).

[48] Y. Shafranovich. *Common Format and MIME Type for Comma-Separated Values (CSV) Files*. RFC 4180. IETF, Oct. 2005 (cit. on p. 57).

[49] Adam Shostack. *Experiences Threat Modeling at Microsoft*. Tech. rep. Microsoft, Jan. 2008 (cit. on p. 28).

[50] Adam Shostack. *Threat Modeling: Designing for Security*. Wiley, 2014. ISBN: 978-1-118-80999-0 (cit. on p. 28).

[51] Remy Spaan. "Secure updates in automotive systems." MA thesis. Radboud University, 2016 (cit. on pp. 25–28).

[52] Dieter Spaar. "Auto, oeffne dich!" In: *c't* 05 (2015) (cit. on p. 5).

[53] Stout Risius Ross. *Automotive Warranty & Recall Report 2016*. Aug. 2016 (cit. on p. 1).

[54] PaX Team. *PaX address space layout randomization (ASLR)*. 2003. URL: https://pax.grsecurity.net/docs/aslr.txt (visited on 12/18/2017) (cit. on p. 7).

[55] Tony Ucedavélez and Marco M. Morana. *Risk Centric Threat Modeling: Process for Attack Simulation and Threat Analysis*. John Wiley & Sons, Inc, May 2015, pp. 317–342. DOI: 10.1002/9781118988374.ch6 (cit. on p. 28).