Toni Celina BSc

Secure Software Remote Update and Diagnosis - Backend

Master's Thesis

Graz University of Technology

Institute of Technical Informatics
Head: Univ.-Prof. Dipl-Ing. Dr.sc.ETH Kay Uwe Römer

Supervisor: Dipl-Ing. Dr.techn. Christian Josef Kreiner
Supervisor: Dipl-Ing. Dr.techn. Gerhard Griessnig

Graz, September 2017

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

Based on the improvements in the automotive industry which resulted in the usage of many electronic devices in vehicles, called electronic control units (ECUs), there is a need to keep the software up-to-date. Since modern vehicles include various functionalities which require an internet connection, updates for ECUs can be transferred over the air. Unfortunately, the automotive industry has not defined standards with regards to security. Due to the lack of security, attackers were able to exploit certain vulnerabilities which harmed the vehicle. The main focus of this project is to detect and cover flaws which were detected by usage of networking technology in the vehicle industry.

Solutions which were developed during the project cover, aspects with regards to the secure update transportation, the key-material management, the reliable communication between a vehicle and the backbone of the system, the update package management, the installation of ECU updates, the gathering of telemetry data from a vehicle, as well as general aspects like usability, and maintainability.

Keywords: firmware update, over the air update, automotive security, connected car, electric control units

Kurzfassung

Basierend auf den Fortschritten in der Automobilindustrie, die eine Anwendung von elektronischen Geräten, welche auch Steuergeräte genannt werden, zur Folge haben, existiert das Bedürfnis die Gerätesoftware auf dem neuesten Stand zu halten. Da viele moderne Fahrzeuge bereits über eine Internetverbindung verfügen, können auch Software Updates für die Steuergeräte über das Internet ausgeliefert werden. Bedauerlicherweise wurden bis heute keine Standards mit Bezug auf Informationssicherheit von der Automobilindustrie definiert. Auf Grund mangelnder Informationssicherheit und Schwachstellen in diesem Bereich sind Fahrzeuge häufig Ziel von Attacken. Der Fokus dieser Masterarbeit liegt in der Ermittlung und Behebung dieser Schwachstellen.

Zusätzlich bietet diese Arbeit Lösungen für sichere Übertragung von Updates, sicheres Schlüsselmanagement, zuverlässige Kommunikation zwischen Automobilen und Infrastruktur, Management von Software Update Paketen, Installation von Software Updates, und Sammeln von Diagnosedaten. Des Weiteren werden auch generelle Aspekte wie die Benutzerfreundlichkeit und die Wartbarkeit der Software behandelt.

Keywords: Firmware Updates, Over the Air Update, Automotive Sicherheit, Vernetzte Fahrzeuge, Steuergeräte

Acknowledgements

I would first like to thank my thesis advisors Dr. Christian Kreiner and Dr. Gerhard Griessnig. Their doors were open whenever I had questions or doubts with regards to the research or the implementation of this thesis. A special thank you goes to Dr. Christian Hanser who supported the project from the beginning until the end of the practical part. I would also like to thank Wolfgang Schöchl who implemented the part with the SmartServiceHub for his cooperation and his contribution to the overall system.

Finally, I must express my very profound gratitude to my parents, my sister, my brother, my girlfriend and to my friends for providing me with unfailing support and continuous encouragement throughout my years of study and through the process of researching, developing and finally, writing this thesis. This accomplishment would not have been possible without them. Thank you.

Contents

[1] Bos et al., 2014.
[2] Housley, 2009.

List of Figures

List of Tables

1 Introduction

Due to the improvements in the automotive industry over the last few decades, vehicles are not only mechanical systems but electromechanical ones as well. The idea to introduce electronic devices into vehicles came in the 1980s, in order to optimize some aspects regarding the fuel-air mixture along with fuel ignition.

As the usage of electronic devices resulted in a significant benefit to the vehicle industry, scientists continued the development of such devices. As a result, today we have cars with more than 50 electronic control units (ECUs). After realizing the real advantage of ECUs, engineers started combining the electronic devices with computer science, which lead to the final outcome - a new type of ECU which can be programmed and calibrated to fit certain requirements of a specific vehicle.

Since the ECUs are integrated into vehicles and there is no other way to update the software but to go to an auto shop, a new way of secure software updates over the air (OTA) was developed in this project.

1.1 Arrowhead

The project developed by this thesis inherited some basic solutions and design decisions from the *Arrowhead*[1] project with regards to the architecture, the security and the communication. Furthermore, this project uses the *Arrowhead framework*[2] by means of communication and authentication.

[1] Arrowhead, 2017(b).
[2] Arrowhead, 2017(a).

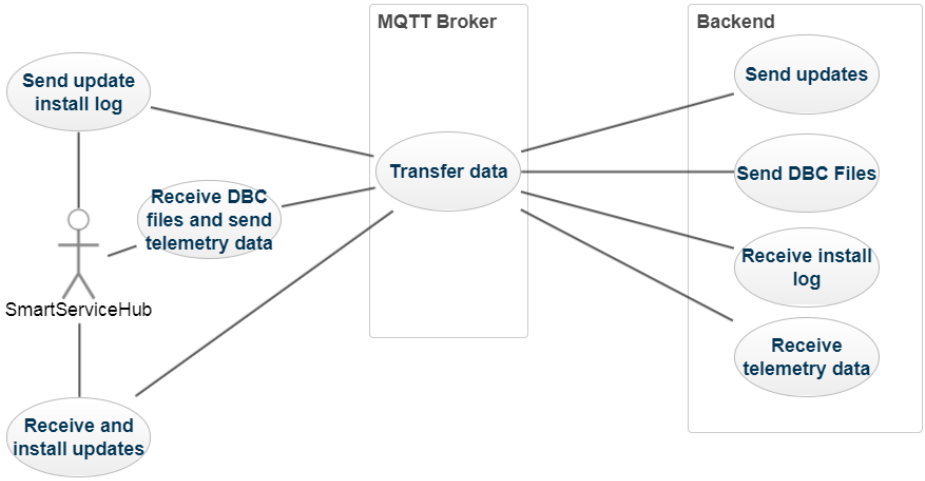


Figure 1.1: Simplified use case and component overview

1.2 Secure Software Remote Updates and Diagnosis

1.2.1 Work

The project work is split into two parts - the frontend and the backend. This thesis will focus on the latter. This includes the development of the backbone of the system, which is able to handle and process the data coming from the frontend and also deploy the firmware packages over the internet.

The two major uses cases for the project are the update of vehicles' ECUs and the gathering of diagnostic data over the air. The figure 1.1 illustrates these two use cases and the architecture which is mentioned below.

1.2.2 System architecture

In order to archive the project goals which are mentioned below, the system is divided into three main components, the *SmartServiceHub*, the *MQTT broker*, and the *backend*. (1.1, 5.1) The *SmartServiceHub* is responsible for

receiving updates, sending update installation logs back to the backend and sending update request in order to receive the newest software from the backend. The *MQTT broker* has the task to forward the data to the recipient. In addition to this task, this component also implements various features with regards to security. The *backend* is responsible for sending updates and processing the incoming data from the *SmartServiceHub*. The components with their respective tasks will be explained in chapter 5 in full detail.

1.2.3 Project goals

Major goals for this project are system security, usability, maintainability, reliability and scalability. Regarding security, state of the art protocols and standards in secure communication are used. Various methods are implemented to ensure the authenticity and integrity of the packets. The same measures are applied to the update packages. The main advantage of the development of such a system is that the needed vehicle updates can now be done remotely. The producers and administrators can send needed updates to all vehicles of a certain type, guaranteeing that the software and firmware are kept up-to-date. Furthermore, in case of a malfunction, the producer can analyze the diagnostic data received through the secure communication methods. This is also beneficial for the owners of the vehicles, as the service updates can be performed over the air. With this achievement, the maintainability and usability goals are reached. For the purpose of this thesis, a set of requirements have been tested. As the system is fine granular, the backend supports the growth and changes in the vehicle market, in such a way that in the case of expansion, no additional adjustments need to be made, only the number of working instances need to be increased so that the payload can be supported.

1.2.4 Verification

The verification process was split into 3 parts - a stress test, a penetration test and a functionality test. The system was tested under a large amount of payloads. The stimuli reached up to several thousands at a time. During

the testing phase, no messages were lost or were not wrongly interpreted by the system. In addition to the stress tests, a penetration test regarding security was made. People from the outside were not able to gain useful insight which would potentially harm the system through cyptographic attacks. Finally, a functionality test was made. The update package, which was sent from the backend, was able to be installed on the given electronic control unit.

2 Motivation

At the very beginning of the automotive industry, vehicles were just mechanical systems consisting of only a few mechanisms using electricity, such as fuel ignition or the lighting system. With the progress made in the electrical industry, the usage of electronic devices and circuits in the automotive industry started to grow, which it continues doing nowadays.

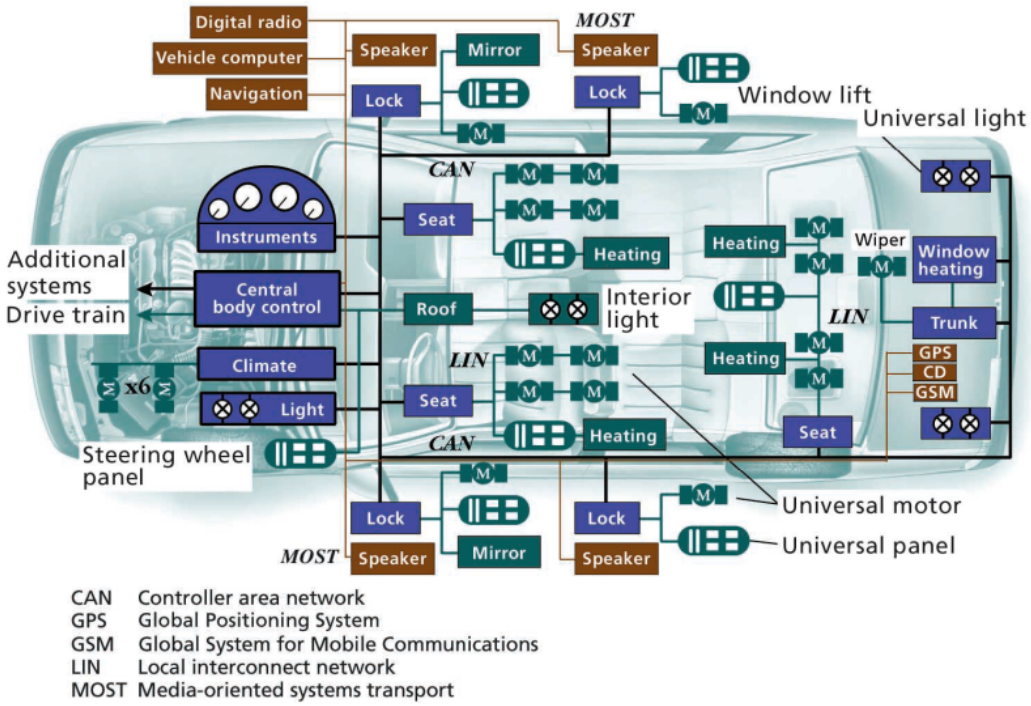
The idea of introducing more electronic devices into vehicles came as experts found a way to optimize some key features in the fuel ignition. They named the device electronic control unit (ECU). Nowadays, vehicles are rather complex electromechanic systems and hold more than 50 ECUs. ECUs have the task to control distinct processes in vehicles, such as already mentioned fuel ignition, brakes, doors, fuel-air mixture regulation, transmission control, assistance systems like ESP, infotainment systems, etc.

There are different classifications of electronic control units. The major classification would be the classification based on the response times - highspeed and low-speed ECUs.

All tasks which are highly critical for the vehicle require a very quick reaction time are handled by the high-speed ECUs. Some examples of such tasks are ignition systems, brakes, transmission control, steering and the main control unit, also known as the central control unit.

All other tasks which do not require such a quick response time are handled by low-speed ECUs. Some of these tasks include mirror control, lighting systems, seat regulation among others.

Since the control units are processing data from distinct actuators, a requirement exists, which exchanges the collected data between different control units. To achieve this goal and connect separate ECUs which have to exchange data amongst each other, a bus is used. Distinct car manufacturers



CAN Controller area network
GPS Global Positioning System
GSM Global System for Mobile Communications
LIN Local interconnect network
MOST Media-oriented systems transport

Figure 2.1: Modern car architecture
Source: Automotive Techis, 2017

and control unit manufacturers use different protocols. The most popular and widely used one is the Controller Area Network (CAN).

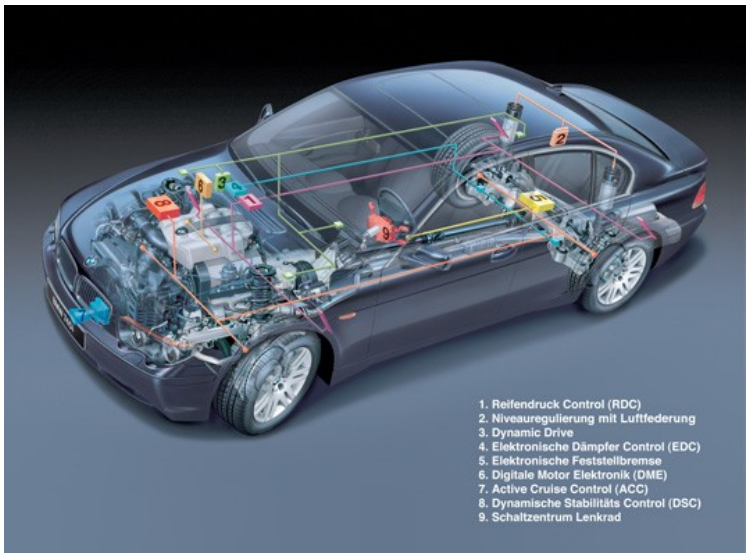


Figure 2.2: Can bus connections
Source: Motorsportzenter, 2017

With the need to connect and exchange information between sensors, actuators, and ECUs, a communication primitive had to be implemented. Since in the time of the expansion of ECUs in automotive industry no suitable protocols existed which satisfied the requirements to connect and exchange data between multiple units, a new protocol had to be introduced. In the act of the leader in ECU development the company Bosch designed and implemented the CAN as a serial bus communication protocol.

CAN is proven to be reliable and efficient, with these characteristics CAN is set as the standard protocol in networked embedded systems. The protocols found its usage in different types of vehicles such as cars, bikes, and trucks. However, the protocol was not limited to vehicles, it was also used in weaving machines, factory automation for medicine equipment and the military industry[1][2]

[1]Xing, Chen, and Ding, n.d.
[2]Johansson, Törngren, and Nielsen, n.d.

Due to the huge expansion of technology usage in vehicles, these vehicles become more exposed to computer attacks. In recent years, attackers succeeded to manipulate some car functions like brakes, wipers, steering and many more[3]. The figure 2.3 shows most components of a modern vehicle, among other things, parts with network access.

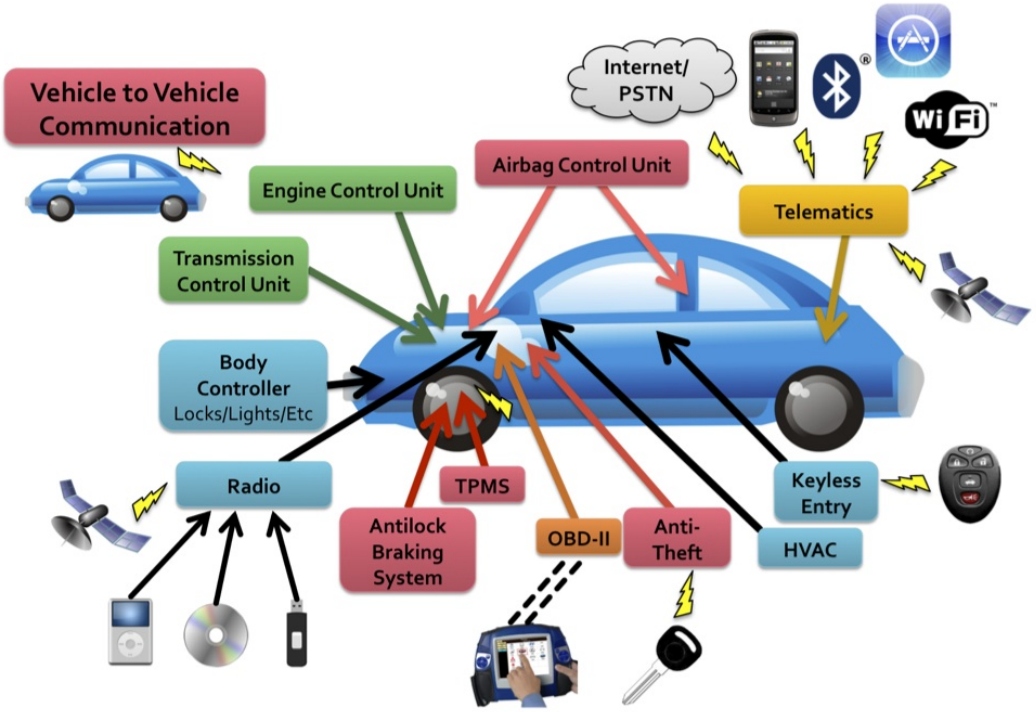


Figure 2.3: Modern car architecture
Source: GlobalCarsBrands, 2017

Since the debut of computer networks in the automotive industry, experts in the field are working on improvements regarding security and safety. However, not only is security a concern, but computer hackers are expanding their focus to breaking into computer networks to steal information and exploit the vulnerabilities of such systems. As the usage of information

[3]Miller and Valasek, 2015.

technology in car industry is expanding, attackers are finding more and more vulnerabilities.

Vehicle producers Tesla[4] and BMW[5] also launched a new technology to update vehicles over wireless networks such as cellular networks and WiFi. This new technology opened the door for hackers to perform even more attacks on vehicles. As the software packages are distributed over the internet, attackers were able to manipulate the software and run malicious code in vehicles.

With the launch of this technology, a new era in automotive industry began. It opened important topics concerning security in vehicles and in the transmission phase during updates.

This master's thesis is focused on the research and identification of potential threats in the usage of networking technology in automotive industry and also the development of a solution covering these topics in order to eliminate the threats and make the car networking as secure as possible.

[4]Tesla, 2017.
[5]BMW, 2017.

3 Background and Related Work

3.1 Research on vehicle security and potential threats

Based on the research and improvements made in the past decades in the automotive industry, the focus has been shifted to security and safety. The outcome of some research topics is already an integrated part of today's vehicles, while other topics are still in the research phase. Based on the research of: insert ref, this chapter will be a brief summary of the automotive industry regarding the used technologies and their vulnerabilities as well as practical attacks in the last few years.

Research on the current technology used in vehicles is done by classifying potential vulnerabilities in different categories based on the access channel to get into the car internal network, the cost to perform an attack, the required knowledge, the impact of the attack and the exposure of the vehicle. Vehicles were inspected either with a direct physical or a remote access with wireless technology usage. Furthermore, the researchers also proved that the attacks are also possible in practice. The following sections will explain the attacks in full detail. In addition to the attack procedure, each section will also explain the damage to the vehicle caused by the attack.

This sections with regards to vehicle security is based on the research Miller and Valasek et. al[1]

[1]Miller and Valasek, 2015.

3.1.1 Physical access

Modern vehicles offer different interfaces to get into the internal car network. The most popular interface for an indirect vehicle access is the On Board Diagnostics II (OBD-II) port which provides direct access to the vehicle's CAN bus. OBD-II is used for programming ECUs as well as vehicle diagnostics. As soon as the attacker is able to observe the traffic from the CAN bus it is possible to perform several attacks. Almost all vehicle manufacturers offer computer-based tools to communicate with the internal car network, these tools are typically in possession of car dealers and mechanics. They are developed to communicate with the CAN bus, both directly and indirectly. Since these mechanisms provide a feature for programming and calibrating ECUs, it is also possible to flood the network with messages which could potentially harm the network and cause unexpected behavior.

Another source of vulnerability is the car's infotainment system. Most of the automobiles today offer a central display with various functions such as the radio tuner, seat heating, car settings, air conditioning system settings and much more. Alongside the central display, nearly all modern vehicles provide interfaces to connect other smart devices with the vehicle.

One of the possible malicious interfaces is included in virtually every vehicle today, the CD player. Researchers found a way to perform an attack on the internal vehicle network by wrapping malicious code into a file format which can be interpreted with an audio player in a vehicle. This format could be any audio format since almost all formats are supported by vehicles' CD players. A practical attack using this interface will be explained in the following section. In addition to the CD player, most of the vehicles offer a possibility to connect a device via USB which could also be used as a backdoor to enter into the car internal network.

3.1.2 Wireless access

In order to simplify actions in the daily routine in automobiles, producers introduced wireless technologies. They applied the technology to keys to make the locking and unlocking of automobiles easier. Some of the keys also

have the functionality to open and close windows. Wireless access features are classified into two categories based on the signal range. Ranges up to 300 meters are considered short ranges and ranges above the 1000 meters mark are considered to be long range signals.

Short range

Due to security risks of using old keys without any kind of identification, the modern industry uses radio-frequency identification (RFID) keys. The RFID key is equipped with a key which is used to unlock the doors and the steering wheel. Many countries across the world set the RFID technology as standard.

Another example for the short range wireless technology usage is bluetooth which is used to connect the car to mobile phones in order to play music or to make phone calls without an interaction with the phone, but with the infotainment system.

Long range

Wireless signals longer than 1000 meters range are considered to be long range signals. Long range signals are distinguished by the number of the receivers. The broadcast signal is received by all vehicles in the network, while addressable signals are received only by the vehicle to which the message was sent. An example for broadcast signals is the Global Positioning System (GPS).

Addressable channels are considered the most vulnerable channel which is exposed by the remote telematics system. The main reason why this type of channel is vulnerable is the fact that an attacker can target a specific vehicle from all over the world and initiate an attack.

Vulnerability class	Channel	Implemented Cabability	Visible to user	Scale	Full Control	Cost
Direct physical	OBD-II port	Plugattack hardware directly into car	Yes	Small	Yes	Low
Indirect physical	CD	CD-based firmware update	Yes	Small	Yes	Medium
	CD	Special song (WMA)	Yes*	Medium	Yes	Medium-High
	PassThru	WiFi or wired control connection to advertised devices	No	Small	Yes	Low
	PassThru	WiFi or wired shell injection	No	Viral	Yes	Low
Short-range wireless	Bluetooth	Buffer overflow with paired Android phone and Trojan app	No	Large	Yes	Low-Medium
	Bluetooth	Sniff MAC address, brute force PIN, buffer overflow	No	Small	Yes	Low-Medium
Long-range wireless	Cellular	Call car, authentication exploit, buffer overflow (using laptop)Call car, authentication exploit, buffer	No	Large	Yes	Medium-High
	Cullular	overflow (using iPod with exploit audio file, earphones, and a telephone)	No	Large	Yes	Medium-High

Table 3.1: Practical attacks
Source: Checkoway et al., 2011

3.1.3 Attack environment and outcome

Based on the theoretical research, the authors of the paper Miller and Valasek, 2015 also made practical attacks on the found vulnerabilities. They classified the attacks based on the vulnerability channel and evaluated the impact of the attacks with regards to the visibility to the vehicle user, the cost to perform the attack, the scale of attack and the type of the access.

The target vehicle was equipped with less than 30 ECUs, connected with multiple CAN buses and linked together. With respect to the connectivity, the car offered following interfaces: bluetooth, CD player, OBD-II port, keyless entry, a satellite radio and the telematics unit as the connector to long range networks. Each attack was performed by generating specific CAN messages and injecting to the CAN bus over a certain interface. Every attack resulted with full access over the target vehicle. Details about the attack results are shown in the table 3.1

3.2 Practical attacks over the past years

This section will provide a brief summary of the practical attacks done on vehicles in the recent years. Some of the attacks are performed as penetration tests in order to improve and point out weaknesses of a given vehicle. However, other attacks are real attacks from hackers which had

major consequences on certain mechanisms. Few vulnerabilities have been eliminated with software/hardware patches from car manufacturers, others are withdrawn for the moment and will be fixed afterwards.

3.2.1 Jeep Cherokee 2014

In 2014, two information technology (IT) security experts Dr. Charlie Miller and Chris Valasek published the paper Miller and Valasek, 2015 after the finished security threat analysis of Jeep Cherokee 2014. They chose the car based on the vulnerability in the infotainment system architecture. The identified threat in the architecture was the direct connection between the head unit and the two main CAN buses.

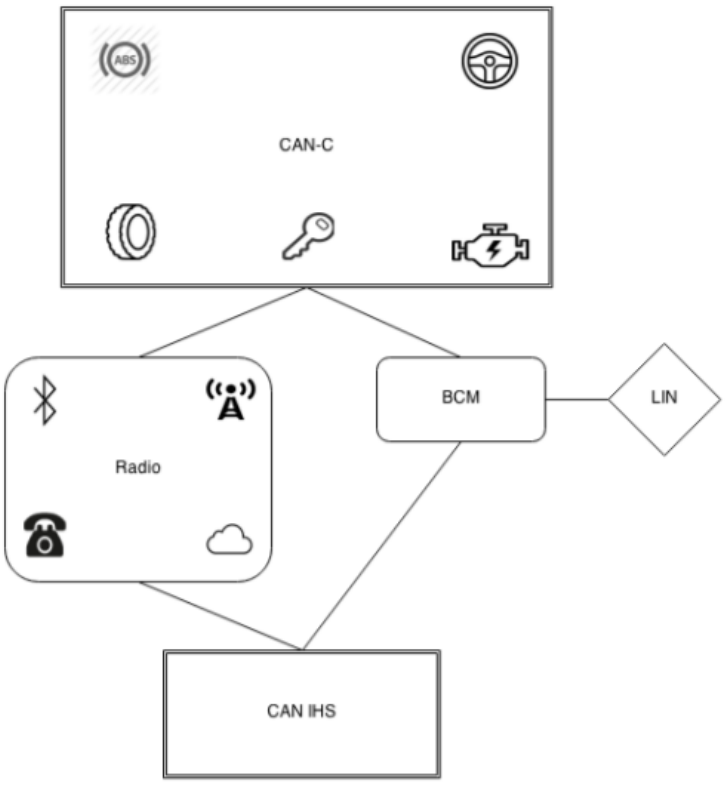


Figure 3.1: Jeep Cherokee architecture
Source: Miller and Valasek, 2015

The actual attacks were performed over three channels, the radio system, WiFi and over a cellular network.

The first approach was to run the malicious software via the UConnect[2] system. The attack was performed using a few simple steps. The first step was to trigger an update of the infotainment system by inserting a USB stick with valid software in order to update the system. The trick concerning the valid software is that the system verifies the update from the USB stick, and only after the update is validated, the system reboots. No further validation is needed and the attacker can insert another USB stick with malicious software and consequently harm the system.[3].

The procedure for exploiting the WiFi access was more complicated. First of all, they had to inspect the function of the WPA2 password generation for securing the WiFi communication. After doing so, they found out that in case that the system cannot get the time, it is automatically set to 00:00:00 Jan 1, 2013, GMT. The next step was to calculate how much time elapsed until the "WifiSvc" generated their initial password which they received from Jeep. In their case, it was 32 seconds, which gave them the ability to brute force the password almost instantly.

In the last approach using cellular networks to harm the system, they came to the conclusion that the D-Bus (Desktop Bus) port was opened in the telematics unit. They estimated the number of potentially vulnerable vehicles between 292.000 and 471.000. In reality, the number of recalled vehicles was 1.400.000.[4]

3.2.2 Toyota Prius and Ford Escape

Earlier in Miller and Valasek, 2014 and Valasek have inspected a Toyota Prius as well as Ford Escape to find possible vulnerabilities in internal car networks. In order to show potential flaws, they developed a tool for communication with the CAN bus. They inspected potential weaknesses of these vehicles by reverse-engineering the communication between two

[2] *UConnect* 2017.
[3] Miller and Valasek, 2015, pp. 34-37.
[4] Miller and Valasek, 2015, pp. 23-27.

ECUs. As a result, they were able to manipulate the CAN message traffic as well as exploit some weaknesses. They investigated the CAN traffic and all vulnerable interfaces in full detail. After that, they were able to take control of the brakes, engine acceleration, steering and also the central display.

3.2.3 Tesla model S

As the leader in today's electric car industry, Tesla is a good example of a state of the art vehicle. Although, a research Dhanjani (2014) showed that the Tesla Model S had vulnerabilities concerning some security aspects. The Tesla company offered an online feature to lock/unlock doors remotely. The main reason why this feature was liable is the fact that the password requirement for such an account was weak. In order to create an account for using those features, a user had to set a password with at least 6 characters including at least one letter as well as only one number. With these requirements, the Tesla website was exposed to brute-force and phishing attacks. The attacker was able to easily get the location of a specific vehicle by using the offered REST API.

Another security threat in the REST API was that Tesla allowed the API usage to third party software. This led to an abuse of user credentials by third party applications. After the release of the article, Tesla advised users not to use these applications. Finally, Tesla released an SDK for third party software in order to get rid of the software bug.

In addition to the vulnerable REST API, the research showed that Tesla also had a weakness concerning the WiFi network. A practical test revealed that some ports like SSH, TCP, HTTP, RPC bind, nfs and X11 were opened.

3.3 Customized products

Due to the fact that an ECU software can be altered and updated afterwards, these kinds of aftermarket products were subject of a research by Koscher in 2015 at University of California in San Diego. One of these products is the Telematics Control Unit (TCU) 3.2.

Figure 3.2: TCU connectivity interfaces (OBD-II, USB)
Source: Foster et al., 2015

After gaining full access to the TCU by performing an attack via physical interfaces, researchers made remote exploits. The TCU 3.2 provided interfaces for SMS as well as an internet interface. By analyzing the code of the TCU, they found out that the web, telnet console and SSH servers were all count to all other network interfaces exept for the USB. As such an architecture included a data modem which was connected to the internet, the attacker would have been able to connect to the TCU by logging into the system with the SSH key which was exploited in previous physical attacks.

Further, they successfully exploited the SMS interface. The TCU was reachable via SMS from the outside by knowing the phone number. Due to the corresponding online documentation of the interface, the researchers found out that one can trigger a remote update over the SMS interface. The logs created after sending SMSs revealed the procedure for remote updating. (See figure: 3.3)

The remote update revealed few significant design failures. First of all, the attacker was able to make arbitrary changes in the update file since the update content was not signed, and therefore the ECU does not check the content for the origin. In addition to the previously found vulnerability, another serious threat in the server was found to be authentication. According to the paper, the server authenticates the device. However, the device does not authenticate the server, which means that the attacker can choose an arbitrary server as the update server.

In order to prove the found vulnerabilities, they approached the problem in almost the same way as in the figure 3.3. The full exploitation process is illustrated in the figure below. 3.4

3.4 Conclusions based on researches in automotive security

The following section will briefly summarize the vulnerabilities researchers have found and point out the weaknesses in modern vehicles over the last few years. Since the introduction of electronic devices in vehicles, vehicles began to be exposed to attacks. Some vulnerabilities are now solved, others are still open, and some are still not discovered. Recently the automotive industry learned a lot about security, building on the experience gathered by solving and detecting flaws in automobiles. Scientists and researchers are now able to solve problems much faster and easier.

3.4.1 Vulnerabilities

The major and frequent shortcomings detected in the recent period will be summed up in the following chapters with regards to the impact, the number of affected vehicles and the actual risk of a certain flaw.

CAN messages and CAN bus

Whether injecting custom CAN messages or modifying existing messages in the CAN bus, the impact on the car internal network is immense. As all of the traffic of ECUs is going through the CAN bus, once an attacker is able to establish a connection to the bus, the possibility to harm the vehicle is immence. As previously demonstrated, there are various ways to access the CAN bus, modify messages and inject malicious messages. Undoubtedly, the easiest way to compromise the CAN bus is an indirect physical connection over an OBD-II interface.

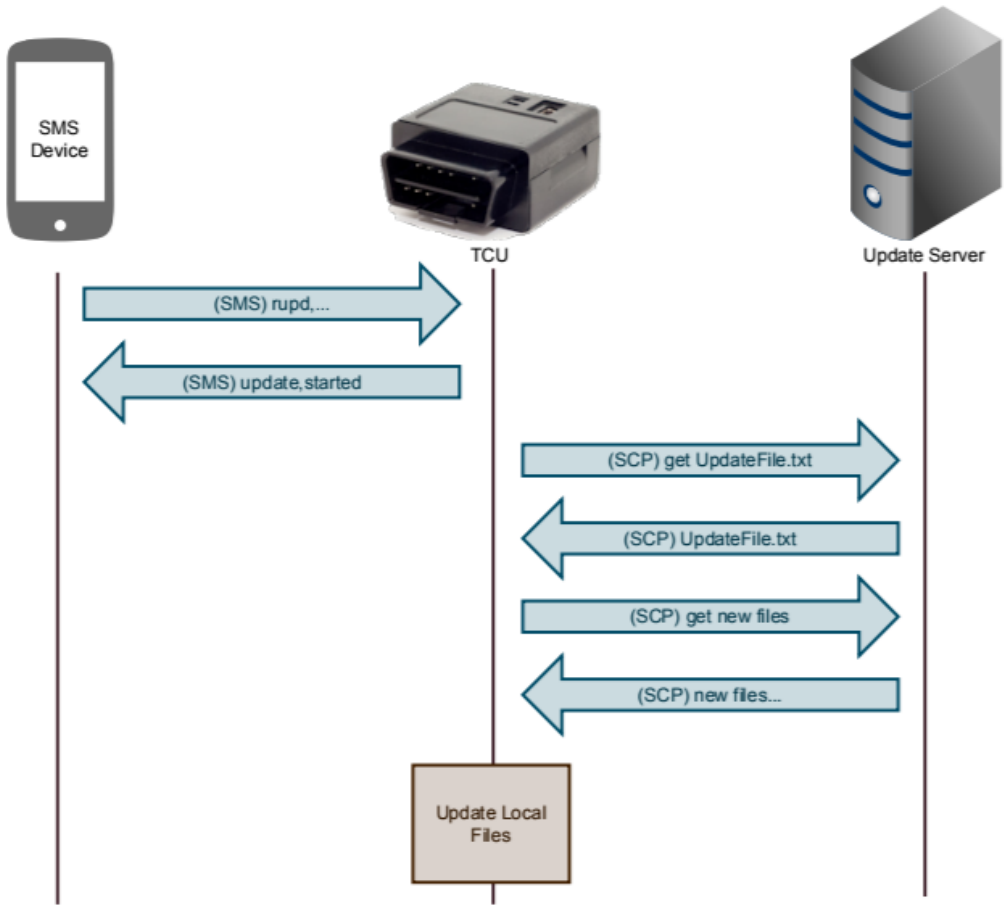


Figure 3.3: Remote update triggered by an SMS
Source: Foster et al., 2015

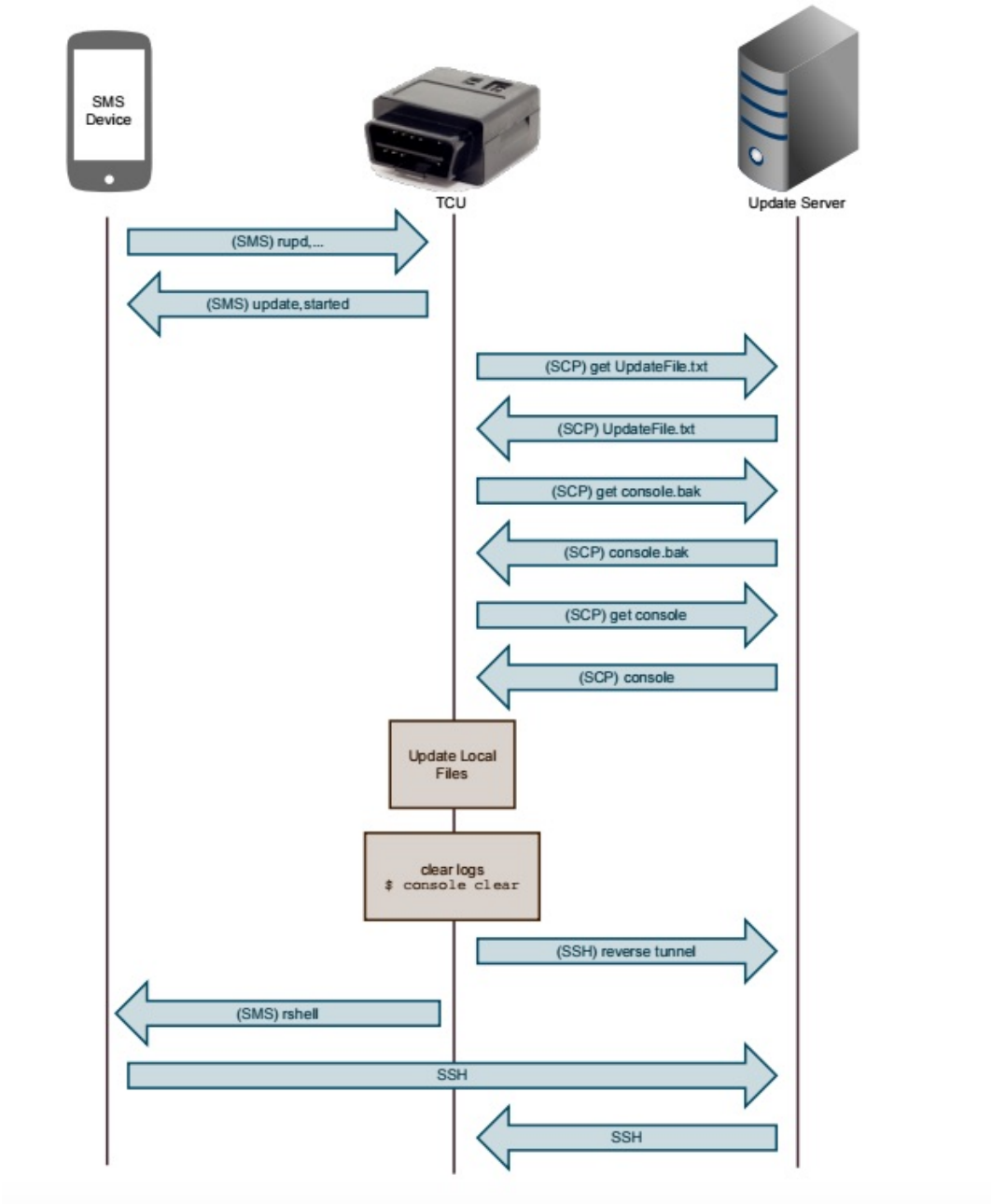


Figure 3.4: Remote update triggered by an SMS (II)
Source: Foster et al., 2015

In all cases, researchers gained almost full access to the vehicle. The affected car functions included the brakes, lighting system, infotainment systems, engine acceleration and the steering wheel.

An appropriate way to prevent attacks on the CAN bus, as well as CAN messages, would be to develop a mechanism to verify the integrity and the authenticity of messages. Since the CAN bus and the ECUs are lightweight devices, with regard to the computing power and the power supply, this seems to be rather impossible.

Key management

As demonstrated in chapter 3.3, researchers were able to enter the network by extracting the SSH key. By doing so, they had full access to the network as well as the possibility to harm the system. The way to get rid of those attacks would be by using a proper key management system. In order to prevent hackers from extracting the key easily, the key should be stored in an encrypted storage. Another approach would be to encrypt the key itself and then to store it. In any case, the best prevention would be not to use the same key for distinct operations; this can be achieved by a key derivation function which guarantees key freshness.

Secure interfaces

Since all attacks are performed over a certain interface, the car manufacturers should put in more effort to secure interfaces which would give access to the internal network. This would not be easy as the main property of an interface is to open a tunnel to the internal network. A possibility to secure interfaces could be a challenge response protocol, which would make the exploits more difficult. A mandatory improvement should be achieved in regards to backdoors which are used to surpass authentication mechanisms.

Authentication and passwords

Chapter 3.2.3 presented exploitations based on weak authentication requirements. The passwords for applications used for accessing vehicles remotely or even directly should be much stronger. In addition to weak passwords, there is also an issue regarding the reuse of passwords, as passwords should be used for one purpose only.

Each feature of the internal network accessed out of the network shall be available only after a certain authentication. In case that a manufacturer provides access to distinct features with the same authentication, a proper hierarchy should be implemented to prevent possible flaws. Passwords should not be shared among people.

Message integrity

As described in chapter 3.3, the origin of messages should be proven to rule out the injection of custom or modified messages to the CAN bus. In case that the message integrity fails, a message could be ignored by the actuators communicating over the CAN bus. Denial of service (DoS) attacks initiated by flooding the network with messages sent from outside the network is also a threat that requires special attention of security experts in the automotive industry. An impact of a DoS attack would result in a catastrophe as the driver would loose control of the vehicle.

One solution to get rid of problems caused by message integrity would be to sign messages, so that the recipient can validate the sender's identity. However, based on the fact that changing a structure of a message by adding a signature would affect the whole protocol, this solution is not applicable. The best possible solution for this problem would be to prevent injecting and altering messages by securing the interfaces.

Secure APIs

Unsecured APIs are a potential threat explained in chapter 3.2.3. If the access to vehicles needs to be provided to a third party software, a certain

security level should be guaranteed by the provider of the access. In order to prevent potential damage, application interfaces have to be as secure as the original software. Another vulnerability found through usage of third party applications was shown to be credentials sharing, which can be mitigated by either using secure applications or not using third party software at all.

Disabling unused ports

Securing main vulnerability sources without securing the neglected opened ports as described in chapter 3.2.3 represents a major threat in the automobile industry. When dealing with long range signals, producers have to make sure that only ports which are needed by a certain service are opened and all other ports are closed. The same precaution measure has to be applied to various debugging ports used for test purposes.

4 Project requirements

Based on the research on automotive security in chapter 4.2, it is clear which aspects require special attention and which aspects have to be covered to ensure a certain level of security. This chapter will describe the key requirements which are needed to reach the project goal of updating ECUs as well as reading telemetry data over the air in a secure way. In addition to security aspects, other important aspects regarding the scalability, the maintainability, the reliability and the reutilization will also be explained.

4.1 General requirements

Alongisde the security requirements which are the most important ones in the project, the project also covered general requisites in order to ensure an easy maintanace, and a high scalability level to support the vehicle market growth.

4.1.1 Usability

One of the most important factors for developing the functionality to remotely update vehicles over the air, as well as collecting diagnostic data in order to monitor the vehicle status and control units behavior, is the usability. The impact of this software is immense since the vehicles do not have to be transported to a vehicle shop to update an ECU, or to inspect the ECU behavior and the status in order to detect and resolve possible malfunctions.

This sotfware also represents a significant improvement with regards to the costs of updating vehicles in comparison to the traditional update routine. The same holds also for running diagnostic services for vehicles, since the software provides the same functionality as the traditional diagnostic service. Now, the same service can be performed over the air, without the physical presence of a certain vehicle.

4.1.2 Scalability

In addition to security benefits of the architecture with three components, the architecture also improves the scalability. In case of a huge expansion in the automotive industry, the project already implements measures to support virtually any number of vehicles. The benefit of the broker component is the fact that the number of such components is not limited, and the incoming message traffic can be distributed not only to one broker, but many.

The same benefits with regards to scalability are provided by the backend as well. The system backbone implements a complex functionality which supplies ECUs with up-to-date software and receives diagnostic data from vehicles. In case that the backend is not able to handle all messages from the broker, or the storage for update packages and diagnostic data runs out, the number of backend components and databases can be extended in order to supply virtually any number of ECUs and vehicles.

4.1.3 Maintainability

As pointed out in Society (2000), the time efforts for maintaining a software range from 65% and 75% of the total development time. Since the time efforts for sofware maintainability are immense, this project has to cover specific aspects in order to save the time invested in the maintanance. According to Kumar (2012), maintainability is the quality aspect to cover following requirements:

- Elimination of latent errors
- Addition of new features/capabilities

- Removal of unused/undesirable features
- Revision of software

Due to continious changes and variations in the automotive industry, a system for remote ECU update has to be easy maintainable and cover the criteria mentioned above. A benefit for the maintainability is the fine granularity of the overall system, and the technology used for the development of every single component. In case that a broker or a backend component needs maintanance for some reasons, the component can be replaced by another one. The replacement, as well as the update of the SmartService-Hub component will be a reseach subject for the future, which will be be discussed in the chapter 8. The software development aspects which cover the mentioned factors will be pointed out in the following chapters.

4.1.4 Reliability

As the software developed by this project is dealing with confidential data, the system has to ensure a high-reliability level. Measures to achieve a reliable system are applied in all stages of development, as well as in all areas of the system. With regards to the communication, reliable protocols have been used. Regarding the content which is sent over the air, cryptographic algorithms which ensure data properties like message integrity, authenticity and confidentiality are used. These aspects will also be discussed in the following chapters.

4.1.5 Reutilization

An important goal of the project was to develop a software which can be used for various purposes. The goal is reached by dividing functionalities into categories and splitting them into units. This was beneficial since the functionalities can be used for many purposes, not only for one.

4.2 General security requirements

This section will provide a brief introduction to the used methods, in order to understand the project goals, as well as the global current state of the used methods in this research area. The techniques which are explained below are there to mitigate threats which are explained in the chapter 3. The main ones include, data authenticity, integrity, non-repudiation, confidentiallity, as well as firewall rules.

4.2.1 A short introduction to cryptography

In order to understand techniques which have been used to reach the goal of this project, a basic understanding of cryptography in modern communication methods is required.

Cryptography in modern communication methods

Since the very beginning of exchanging information, there was a need to prevent others from accessing confidential data. To make sure that only two parties which are exchanging some piece of information can actually access the data, cryptographic systems were designed.

The core of every cryptographycal system are mathematical functions. The key to the usage of those functions is the fact that they are hard to compute, even for computers. These functions are used by algorithms for various purposes like the encryption, the decryption, the key exchange and the key generation. To ensure that only the recipient can read certain data, encryption is used. Encryption is the operation of encoding data in a way that only authorized individuals can decode and access the data. In the other hand, the reverse process of decoding data is called decryption. In most cases, to encode as well as decode a piece of information keys are used. Keys are a sequence of alphanumerical characters whose length depends on the encryption/decryption algorithm which uses the key.

4.2.2 ECC Cryptography[1]

Ecliptic curves which are most commonly used in real-world problems have the Weierstrass form form $E : y^2 = x^3 + ax + b$. They are defined for a finite field $\mathbf{F}_p$, where $p > 3$ is prime and a; b $E \in \mathbf{F}_p$.

The cryptographic group which works on protocols is given a curve E, who's form has been defined above. This group is a large prime-order subgroup of the group $E(\mathbf{F}_p)$ of $\mathbf{F}_p$-rational points on E. This group consisting of rational points, has all solutions of $(x; y) \in \mathbf{F}_p^2$ to the curve equation together with a point at infinity, or the neutral element The amount of $\mathbf{F}_p$ - rational points is denoted by $\#E(\mathbf{F}_p)$ and the prime order of the subgroup by n. The based point is denoted by $G \in E(\mathbf{F}_p)$, and is a fixed generator of the cyclic subgroup.

NIST recommends using five elliptic curves for the elliptic curve digital signature algorithm, which targets five different security levels. Each curve is defined over a prime field, which is a generalized Mersenne prime. The same coeffiecient $a = 3$ is used by all curves, with the goal of this being efficienty. The five recommended primes include

- $p_{192} = 2^{192} - 2^{64} - 1$
- $p_{224} = 2^{224} - 2^{96} + 1$
- $p_{256} = 2^{256} - 2^{224} + 2^{192} + 2^{96} - 1$
- $p_{384} = 2^{384} - 2^{128} - 2^{96} + 2^{32} - 1$
- $p_{521} = 2^{521} - 1$

As mentioned above, efficient endomorphisms is used for the speed up scalar multiplication, but what was not said is that it also speeds up the computing of discrete logarithms. One the other hand, the automorphism group of E has the order 6. An elliptic curve with j-invariant which is not 0 and 1728 only has an order 2, such that the speed-up in Pollard's rho algorithm is a constant factor.

Also important to note is for the larger automorphism group is the existence of six twists. It is needed to pay extra attention to the curve's twist security for an implementation using x- coordinate only arithmetic. In other words,

[1] Bos et al., 2014.

its quadratic twist needs to have a large enough prime divisor for the discrete logarithm.

This prevents from an attacker gaining multiples with secret scalars of a point on the quadratic twist. The quadratic twist of secp256k1 has a 220-bit prime factor and is therefore considered as twist secure .

Elliptic Curve Public-Key Pairs

Suppose that a set of domain parameters is given, which includes choices for the base field prime p, the elliptic curve $E/\mathbf{F}_p$, and the base point G of order n on E. An elliptic curve key pair (d, Q) consists of a private key d, which happens to be a non-zero modulo the group order n which is selected at random, in addition to a public key $Q = dG$, the d-multiple of the base point G. Subsequently, the point Q is also selected randomly, and is within G.

Elliptic Curve Key Exchange

Two parties need to agree on a shared key, and each party individually generates key pairs - pairs (d_a, Q_a) and (d_b, Q_b). Next, the parties exchange the public keys Q_a and Q_b, this is done in a way in which it is possible to calculate the point P as a function of the two, $P = d_aQ_b = d_bQ_a$. The shared secret key is found by using a key derivation function.

Elliptic Curve Digital Signatures

The signer makes a key pair (d, Q), which includes a private signing key d and a public verification key $Q = d_G$. In orger to sign a message m, the signer needs to choose a random integer k such that $1 \leq k \leq n - 1$, which varies per message, it is important to note that k must not be revealed. Next, the signer calculates the point $(x_1, y_1) = kG$, transforms x_1 to an integer and computes $r = x_1 \ mod \ n$. The message is then hashed to a bitstring, which has the length which is no longer than the bit length of n, which is then transformed to an integer e. The signature of the message is the pair (r, s);

(where r and s both need to be different than zero) of integers modulo n, where $s = k^{-1}(e + dr)\ mod\ n$.

Beforehand it is mentioned that k must be kept a secret, this is because otherwise the secret signing key d can be computed by $d \equiv r - 1(ks - e)$ $(mod\ n)$, as r and s are given in the signature and e can be computed from the signed message. Moreover, in the case that the same value for k is used to sign more than one message, for example it is used for two messages, the same signing key d is used and it produces signatures (r, s_1) and (r, s_2). From this information, it is easy to find the value of k, which is defined as $k \equiv (s_2 - s_1)^{-1}(e_1 - e_2)$ $(mod\ n)$, which then would allow for the recovery of the secret key.

4.2.3 Secure communication protocol

In order to prevent threats which are summarized in the chapter 3, the commmunication have to ensure a high level of security. The main security properties which have to be covered by the project will be explained in the following sections.

4.2.4 Key Management

As the communication is encrypted with an end-to-end encryption, there is a need to securely store the key material[2]. In order to get rid of common problems with the storage of key material, the database which stores private and public keys is located in a secure demilitarized zone (DMZ)[3][4]. The communication details with regards to the encryption and the storage of the key material will be explained in the section 5.2. In order to store the key material, a database is used. The database manages key for both scenarios, for the data encryption/decryption, as well as certificates for broker authentication (See 5.3.2). Details regarding the storage of the key

[2]Bellovin and Housley, 2005.
[3]Cheminod, Durante, and Valenzano, 2013.
[4]Mazur et al., 2016.

material, as well as for the storage of data in general will be explained in the section 5.3.3.

4.2.5 Update package management

Since the ECUs update process deals with confidential firmware updates for vehicle ECUs, there is also a requirement to store them in a secure way, in order to prevent potential flaws with regards to data modifications which are pointed out in previous chapters. As the requirement for update package storage can vary for different vehicle manufacturers, there are two covered scenarios in regards to the data which is stored in the database. The basic scenario is implemented for the case that a firmware update is not digitally signed[5] by the manufacturer and the other case when the firmware is digitally signed by the manufacturer. Both cases will be explained in the following chapters.

4.2.6 Update package security properties

In order to get rid of problems with regards to message modifications which are a common problem pointed out in the chapter 3 the update package has to ensure some properties, such that, the vehicle can justify the origin of the update package and that the vehicle can be sure that the package was not modified during the transport.

This section will focus on the general goals of cryptography and will try to explain in a simply manner how cryptography works. Although, generally speaking, cryptography has a large number of goals, there are considered to be four prime ones, from which all others can be derived. These four include confidentiality, integrity, authenticity and non-repudiation.

[5]Menezes, Oorschot, and Vanstone, 1996.

Confidentiality

Confidentiality, or privacy, means that the information content is kept secret, or only know to and kept for the parties which are authorized to have it. The privacy of the message can be kept in various ways, some methods can include actual physical protection, or the use of complex mathematical functions and algorithms which could hide the true meaning of the content. (Menezes, Oorschot, and Vanstone (1996))

Authenticity

Authentication is not only applicable for the data itself, but also to the entities involved in the communication. This means that the data should be authenticated, with regards to its origin, the date of its origin, the content, and so on. Furthermore, as mentioned previously, all parties involved have to be authenticated, to make themselves known and be identified. As authentication applies to two different aspects, there is a split between data origin authentication and entity authentication. (Menezes, Oorschot, and Vanstone (1996))

Data integrity

Data integrity focuses on keeping the given data accurate and consistent during its lifetime, or in other words it focuses on the unauthorized modification of the data. For this goal to be realized, one must be able to notice changes in and manipulation of data by parties which do not have authentication to do so. (Menezes, Oorschot, and Vanstone (1996))

Non repudiation

Non-repudiation forbids a party from declining previous commitments. In case that some conflicts of opinions arise with regard to an entity denying that some actions were taken, and in these particular cases a third party is

needed to find a solution to the problem. (Menezes, Oorschot, and Vanstone (1996))

4.3 Backend security

In addition to general security requirements, one has to define and analyze the backend security in order to design a highly secure system. The approach was to analyze the threats with regards to the backend security using Microsoft STRIDE[6] in order to protect the system against them.

This section will describe the current threats and countermeasures with regards to the backend security as the focus of this project lies in the development of the backend. Since the backbone of the system provides various services, some of which include the storage of updates, the key management, the encryption/decryption of the content which is sent over the network, and primitives which provide the access to the database which contains confidential data, potential threats and countermeasures will be explained in more detail below.

4.3.1 Threat categories

In order to secure applications against attacks, one has to identify the threats which are relevant for a specific application. A threat could be any malicious element which can harm an application.

According to STRIDE, threats are classified into the following categories:

- *Spoofing*, an attempt to access the system using a false identity. Accomplished using stolen credentials, or using false IP[7] addresses.
- *Tampering*, unauthorized data modification
- *Repudiation*, ability to deny that a certain action is performed by a specific user
- *Information disclosure*, undesirable exposure of private data

[6] Microsoft STRIDE, 2017.
[7] J. Postel, 1980a.

- *Denial of service*, making a service unavailable
- *Elevation of privilege*, using a privileged user identity to gain privileged access to the system

Threat	Countermeasures
Spoofing user identity	Use strong authentication Do not store secrets (for example, passwords) in plaintext. Do not pass credentials in plaintext over the wire. Protect authentication cookies with Secure Sockets Layer (SSL)
Tampering	Use data hashing and signing. Use digital signatures. Use strong authorization. Use tamper-resistant protocols across communication links. Secure communication links with protocols that provide message integrity.
Repudiation	Create secure audit trails. Use digital signatures.
Information disclosure	Use strong authorization. Use strong encryption. Secure communication links with protocols that provide message confidentiality. Do not store secrets (for example, passwords) in plaintext.
Denial of service	Use resource and bandwidth throttling techniques. Validate and filter input.
Elevation of privilege	Follow the principle of least privilege and use least privileged service accounts to run processes and access resources.

Table 4.1: Proposed solutions
Source: Meier et al., 2003

This section will summarize network threats based on the Microsoft[8] research (Meier et al., 2003), regarding the backend, which is often an object for attackers. Furhermore, the attacker targets the network infrastracture in the interest of exploiting badly configured network devices, the main ones include routers, firewalls and switches.

As for threats in general, STRIDE also classified threats into different categories which are listed and explained below.

- *Information gathering*
- *Sniffing*
- *Spoofing*
- *Session hijacking*
- *Denial of service*

[8]Microsoft, 2017.

Information gathering, the usual procedure to gather information through the network is to identify open ports with port scanning for the sake of finding out information about the operating system, applications, and application versions. Consequently, with all this knowledge gained, the attacker could then attack known flaws and harm the system. The usual countermeasures as to the preventention of such attacks, consist of configuring routers in the way that they block footprinting requests, as well as disabling unused ports which could be misused by attackers.

Sniffing is a process where the attacker monitors the network traffic with the goal of finding information which can be used for a more dangerous attack. Generally, the attacker is looking data concerning passwords or configuration information. As in all networks, this data can be encrypted or not. In case that the network traffic is encrypted by a lightweight encryption, the attacker can use algorithms to decode the ciphertext, otherwise, in the more dangerous case, the data is sent as plaintext, and therefore the decyption process is not needed. Solutions to such problems are the usage of strong physical security, prevention of gathering information locally, and the usage of strong data encryption.

Spoofing is a technique which is used to hide a true identity in the network. In the attacker's point of view, spoofing is very beneficial since it hides the source of the attack. Even though carefully created spoofing packets may never be detected by the user, there are some countermeasures against spoofing. In order to get rid of spoofing attacks, one possible solution would be to filter incoming packets which appear to come from an internal or an invalid local IP address.

Session hijacking, or man in the middle attack, forces a client or a server to believe that the attacker is the legitimate host. This is done since the man in the middle can manipulate data in the traffic to the actual end point. Solutions to this problem include the usage of encrypted session negotiation and the usage of encryption in general.

Denial of service prevents the user to access the services of the server. This kind of attack is very dangerous since it is easy to perform the attack, and also very difficult to track it. The actual attack uses potential vulnerabilities

in the TCP/IP[9] stack. The countermeasures involve the hardening of the TCP/IP stack, as well as the usage of intrusion detection systems in order to identify and respond to SYN[10] attacks.

4.3.2 Host Threats

In addition to common network attacks and threats, STRIDE classifies the host threats as follows:

- *Viruses, Trojan horses, and worms*
- *Password cracking*
- *Denial of service*
- *Arbitrary code execution*
- *Unauthorized access*

Viruses, Trojan horses, and worms are special programs developed to perform malicous operations on the host system. In the interest of preventing the installation and execution of such programs, one should block all unused ports at the firewall and at the host, update the operating system, and disable all unnecessary services and protocols.

Password cracking is a common technique with the goal of getting the access to the server. Countermeasures against this specific threat are the usage of strong passwords, as well as the frequent change of passwords. Other measures include lockout policies which prevent the logging in after a certain number of failed login attempts, and the analysis of failed login attempts in order to find patterns which are used for password hacking.

Denial of service, in addition to the previous section with the DoS attack on the network, the same attack can also be applied to the host. The attacker can try to harm the host by brute forcing against the application on the host, or try to exploit vulnerabilities which exist in the service or attack the operating system which runs on the host server. Along with countermeasures for DoS attacks on the network, the host should be capable to handle high traffic payloads.

[9]Jon Postel, 1981.
[10]Eddy, 2007.

Arbitrary code execution is a very dangerous threat where the attacker can execute his own code on the host. Typical flaws include a weak firewall, buffer overflows at the host server and servers which allow path traversal. To get rid of these problems, one should prevent URLs[11] with "../" to avoid the path traversal flaw and stay up-to-date with updates and patches to prevent buffer overflow attacks.

Unauthorized access is caused by a lack of access control allowing attackers to use the system without authorization. The precaution measures to prevent unauthorized access include proper web access permissions and the lock down of files with file systems permissions.

4.3.3 Application threats

In order to protect the application from the common flaws, one has to analyze the threats in favor of protecting the application gainst them. According the Microsoft (Meier et al., 2003), the most common threats are the following ones:

- Input validation
  - Buffer overflow
  - SQL injection
- Authorization
  - Elevation of privilege
  - Disclosure of confidential data
  - Data tampering
- Sensitive data
  - Eavesdropping
  - Data tampering
- Cryptography
  - Poor key generation or key management
  - Weak or custom encryption

[11] Berners-Lee, Masinter, and McCahill, 1994.

Input validation

In case that the application performs an input validation, an attacker can execute an attack by carefully preparing the input in such a way that the application assumes that the input comes from a valid source. Additionally, when the host and the network are secured against such attacks, the application becomes the attacker's target.

The most common techniques for such attacks are a buffer overflow attack, which can lead to the denial of service and the SQL injection attack which can be used to fetch certain data from the database. SQL injection can be performed when the application uses dynamic query generation, as well as when the input is not previously validated. In general, the attacks can be mitigated by performing a proper input validation. In addition to the input validation, the SQL injection can be avoided with stored procedures, alongside with the usage of prepared statements instead of dynamic queries with concatenated parameters.

Authorization

Typical attacks regarding the authentication are the elevations of privileges, the disclosure of conditional data and data tampering. The privilege escalation can be performed when the application authorization model is weak and poorly designed. Also, the attacker can gain access of the higher privileged user and in this way, retrieve confidential data. Similar to privilege escalation, the attacker can gain the access to confidential data when the application does not require any authorization, this can be mitigated by role checks and the by the usage of encryption when storing data. Another common flaw in the application layer is data tampering, or in other words, the modification of data. This can be avoided by strong access control and role-based security.

Sensitive data

Since the project deals with sensitive data, vulnerabilities like eavesdropping and data tampering have to be avoided. A common weakness known as network eavesdropping occurs when the data is transmitted over insecure channels like HTTP[12] without encryption. A mitigation against eavesdropping is to transmit the data over secure channels like SSL/TLS or to encrypt the data before transmission. Another typical threat is the tampering of data, which refers to the modification of the data when transmitted over a network. This threat can be avoided by signing the data with for example HMAC[13].

Cryptography

Common flaws which refer to cryptography are weak or custom encryption and the poor key generation or key management. The first threat can be avoided by using standard proved encryption methods. Poor key generation can be mitigated by using strong random key generators and the key management threat can be avoided with secure key management and key expiration.

4.3.4 Summary of backend security threats and countermeasures

As described in previous sections, security methods have to be applied in all areas in order to avoid potential data loss and/or data modification. Furthermore, this research concerning backend security pointed out the common vulnerabilities which require special attention by the project development. The common threats are listed below. In addition to the summary of the typical threats to backend security, countermeasures are also proposed.

- *Viruses, Trojan horses, and worms*

[12] Fielding et al., 1999.
[13] Krawczyk, Bellare, and Canetti, 1997.

- *Password cracking*
- *Arbitrary code execution*
- *Unauthorized access*
- *Information gathering*
- *Sniffing*
- *Spoofing*
- *Session hijacking*
- *Denial of service*
- *Cryptography*
- *Authorization*
- *Input validation*

As the projects' task is to transmit updates for vehicles over the air, as well as collect data from vehicles, these countermeasures have to be applied in order to achieve a high level of security. The way in which the countermeasures are applied will be explained and summarized in the chapter 5, and also shown in the table 5.2. Furthermore, the most important system components and informations, offten called assets[14], which have to be protected are listed below.

- *Update packages*
- *Diagnostic data*
- *Key material*
- *Vehicle data*
- *Update installation logs*
- *System components* (5.3)

[14] *Management of information and communications technology security – Part 1: Concepts and models for information and communications technology security management* 2004.

5 System Architecture

The system is divided into three main parts. The backend and the SmartServiceHub are connected to a central party which is resposible for transfering messages to the other parties. This central party acts as a broker which receives and forwards messages to the final recipient. All three actuators in the system are equipped with various security and safety mechanisms to fit the requirements which are mentioned in the previous chapters.

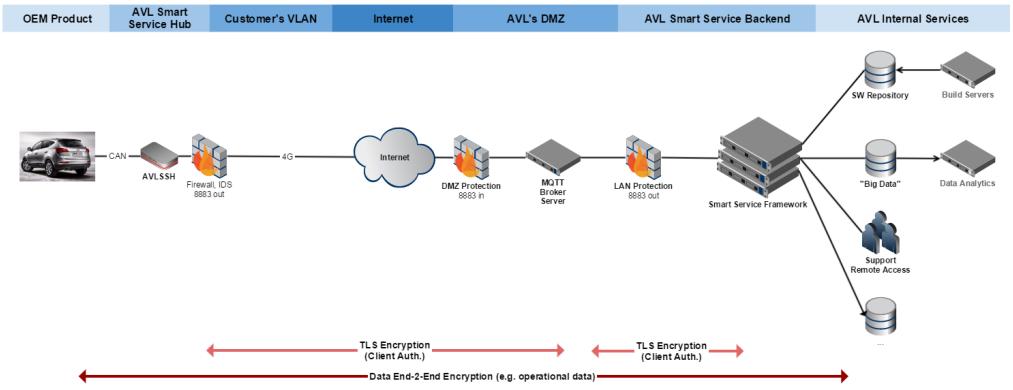


Figure 5.1: System Architecture

5.1 Design decisions

This chapter will briefly explain design desicions which were made to reach the project goal. As this thesis is based on the findings of the Arrowhead project[1], this chapter will also explain the most important discoveries of

[1] Arrowhead, 2017(b).

Arrowhead, which were used by this project.

5.2 Communication

The following sections will describe solutions which cover the requirements from the chapter 4.2.3 with regards to the secure communication. As it is illustrated in the figure 5.1, the data which is sent to the broker is encrypted by an authenticated encryption (see 5.2.2), and aditionally secured by the Transport Layer Security (TLS)[2]. Furthermore, the architecture also covers the threats which are explained in sections 3.1 for the front-end and 4.3 for the backend, while this projects' work is focused on the latter. Threats which are covered by this architecture will also be summarized in the table 5.4

5.2.1 TLS

The main goal of the TLS protocol is supply a structure for a secure communication between the two parties included. This is established through a network, for which neither of the parties has an end-to-end control, giving a thirst party the chance to intercept the communication. Below, the two main features, data integrity and end-point verification, will be described in greater detail.

TLS data integrity

Data integrity has already been described. To summarize, the focus is to keep the data from being modified by unauthorized parties. In other words, the communication between the entities is done securely, meaning that, the data is received without any alterations and without the possibility of having other unauthorized parties accessing this communication. To help ensure data integrity, TLS uses various techniques, including asymmetric

[2]Dierks and Rescorla, 2008.

and symmetric encryption. These techniques prevent a third party from accessing the content, even if it is able to intercept the message. As previously mentioned, it also protects the messages from any modifications, including removing line, inserting them, etc. (Horman, 2005)

Endpoint Verification

End-point communication was briefly mentioned in the form of "entity integrity" beforehand. As important as it is to verify the integrity of the data, it is equally important to do the same with regards to the parties participating in the communication. Basically, the involved entities or endpoints are who they claim to be. TLS checks this with the use of end-point certificates. When a TLS connection is established, message is signed with the entity's certificate and is sent along with the certificate. To check if the certificate is valid, several checks can be performed. Naïve checks include making sure that the certificate has not expired and also checking if the names of the hostname match the one of the common name which it is connected to. Further checks include seeing if it is signed by a known certificate authority, which are often found in a list within the browser. (Horman, 2005)

5.2.2 CMS[3] Encryption

Hybrid encryption uses the benefits of symmetric-key and public-key cryptography systems, as it is as efficient as the first and as convenient as the latter. Symmetric-key systems are known to be more efficient when it comes to encrypting messages, as they use a shared secret message between the two parties communicating the message. On the other hand, public-key systems require the use of complex mathematical operations, which cause these systems to be less efficient in comparison to the symmetric-key. However, they do not use a shared secret and are, for this reason, considered to be more convenient. As mentioned beforehand, symmetric-key systems use a shared secret message. A key transportation scheme is needed in order to

[3]Housley, 2009.

send it from the originator to the receiver of the message and it includes the key agreement scheme, as well as the key-wrapping algorithm. These will be described in more detail later on.

Elliptic Curve Menezes-Qu-Vanstone key exchange

Elliptic Curve Menezes-Qu-Vanstone (ECMQV)[4] is a one-pass key agreement scheme who's goal is to create a shared secret between the two parties, which would then be used to encapsulate a session key. This means that it is a protocol which can be executed by the originating party, without actually having any contribution from the receiver. Suppose U represents the originating party, whereas V represents the receiving party. The one-pass scheme C(1e, 2s ECC MQV) includes the short-lived contribution by the originating party U, as well as the long-term key pairs for both parties involved and the ECMQV primitive ecmqv(). It is necessary that both parties come to an agreement concerning the Elliptic Curve (EC) domain D = (q, FR, S, a, b, P, n, h) and a key derivation function (KDF) kdf(). Below, the steps of a one-time setup of long-term keys:

- 1) U: static private/public key pair (d_{sU}, Q_{sU}) in D.
- 2) V : static private/public key pair (d_{sV}, Q_{sV}) in D.
- Process for U: $(Q_{eU}, k_U) = kas_U(Q_{sU}, d_{sU}, Q_{sV})$
- 1) Generate random ephemeral key pair (d_{eU}, Q_{eU}) in D.
- 2) Compute $Z = ecmqv(d_{sU}, Q_{sV}, d_{eU}, Q_{eU}, Q_{sV})$.
- 3) Derive $k_U = kdf(x_Z)$, where x_Z is x-coordinate of Z.
- 4) Output shared secret k_U and Q_{eU} .
- Recipient V calculates $k_V = kas_V(Q_{sU}, d_{sV}, Q_{eU})$ with its static private key, and the ephemeral Q_{eU} obtained from U:
- 1) Compute $Z = ecmqv(d_{sV}, Q_{sU}, d_{sV}, Q_{sV}, Q_{eU})$.
- 2) Derive $k_V = kdf(x_Z)$, where x_Z is x-coordinate of Z.
- 3) Output shared secret k_V .

At this point, both parties involved have obtained the shared secret $k_U = k_V$.

[4] Blake-Wilson, Brown, and Lambert, 2002, Chapter 4.

Key wrapping schemes use a symmetric-key system in order to maintain confidentiality and integrity for the storage and transport of material which is needed for the generation of the secret key. A key wrapping scheme consists of a function for wrapping and unwrapping, which use a secret key wrapping key, the secret key material, as well as the protected wrapped key material. The wrapping function is used for the encryption of the secret key wrapping key the material. On the other side, the unwrapping function is used to decrypt the secret key wrapping key and the protected wrapped key material. It also is used for the verification of the authenticity of the received secret key material.

The main goal of Authenticated Encryption (AE)[5] is to accomplish data confidentiality and authenticity. These schemes are semantically secure, which means that attacker is not able to gain any information about the plaintext from the ciphertext. Moreover, they are secure against given cipher text attack. The reason for this is because AE can detect invalid ciphertext, and therefore it can refuse to decrypt it. As key wrapping schemes, AE schemes also consist of encryption and decryption functions. The encryption function $authenc\_k()$ figures out the ciphertext, using the plaintext as well as the key. Furthermore, an authentication tag of the plaintext, the associated data (optional, not encrypted), and the key is generated. On the other hand, the decryption function $authdeck()$ decrypts the ciphertext and chekc the authenticity of the message using its key. In case that any of the following: the plaintext message, the associated data or the authentication tag are modified, the decryption process will terminate in failure, since the integrity has been violated.

Authenticated encryption

In the paper Lesjak, Bock, et al., 2016, the researchers describe a snapshot protection system which protects the snapshot between different parties – the mediator and customer or mediator and vendor, regarding confidentiality, integrity and authenticity of the transferred snapshot.

[5] **cryptoeprint:2003:069**.

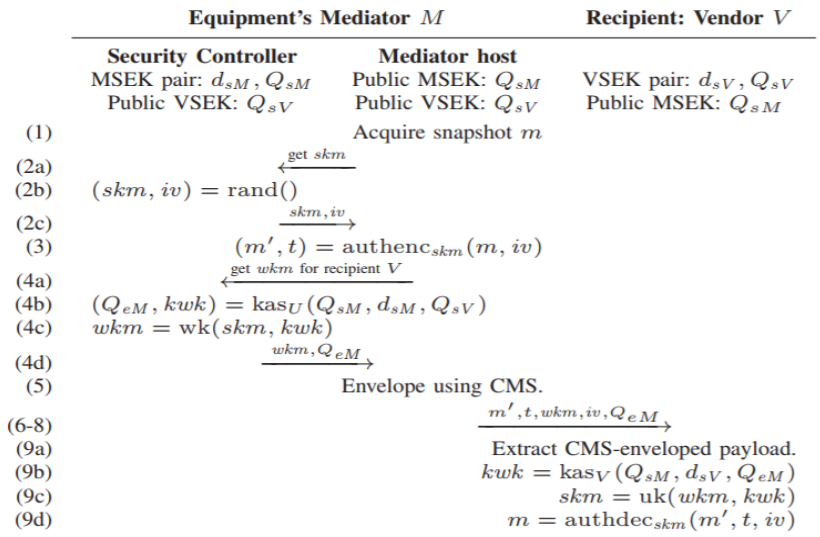


Figure 5.2: Encryption
Source: Lesjak, Bock, et al., 2016

The figure below 5.2.2 shows the main steps of the protocol between the originating party (in this case – the mediator), and the receiving party (here, the customer or the vendor), in the form of a table. The individual steps will also be explained in more detail. The broker is disregarded because as it does distribute the protected snapshots, it is not able to decrypt the given snapshots, and is therefore irrelevant for this specific case. The hybrid encryption scheme is based on the ECMQV, which was described previously. Furthermore, the partitioning of the originator's steps split among two execution environments- a secured one and a general-purpose one can be seen.

As already mentioned, the steps from the table are explained in more detail below.

- Steps 1–2c: in the case that the originator (ie. the mediator) needs to transfer a snapshot to the receiver (again, in this case, the vendor), a request for a short-lived, ephemeral, secret key material (SKM) and an initialization vector (IV) from the SC is made. The values are generated by the SC, as its TRNG provide a cryptographically qualified entropy

d_{sX} , Q_{sX}	Static (long-term) private and corresponding public key of X.
d_{eX} , Q_{eX}	Ephemeral private/public key pair generated by X.
skm	Ephemeral secret keying material to encrypt/authenticate m.
m	Plain-text message (snapshot) to be transferred to a recipient.
m'	Encrypted snapshot m.
t	Tag value to verify authenticity of encrypted message m'.
kwk	Ephemeral key wrapping key obtained from kasU or kasV.
wkm	Wrapped key material containing encrypted skm using kwk.
$rand()$	Random number function to generate ephemeral key material.
$kas_X()$	KAS to calculate kwk for X.
$kas_V()$	C(1e, 2s, ECMQV)-type KAS to calculate kwk.
$authenc()$	AE scheme to encrypt and authenticate.
$authdec()$	AE scheme to decrypt and verify.
$wk()$	Wraps and encrypts skm using kwk.
$uk()$	Unwraps and decrypts wkm using kwk.

Table 5.1: Encryption - terms explanation
Source: Lesjak, Bock, et al., 2016

source, in comparison to the mediator's host CPU.

- Step 3: The MQTT plaintext payload is protected with the help of authenticated encryption. The goal of this is end-to-end data confidentiality and authenticity, which is described in the section above. To recall, the encryption function authenc() is used to protect the payload with the help of the short-lived SKM and an IV. It delivers an authenticity tag value, which is needed to verify the snapshot's authenticity and integrity to the other parties involved in the communication.

- Steps 4a–4d: The originating party makes a request for the SC to wrap the secret key material for the receiving party. This is done by the SC performing the C(1e, 2s, ECMQV) type key agreement scheme (KAS), using the originating party's snapshot encryption key (MSEK) pair, alongside the recipient's public snapshot encryption key (VSEK), as the input. It is important to note that the SC does not receive the wrapped secret key material from the originator's host, but takes it from the preceding step (step 2b). In this way, an attacker is prevented from wrapping arbitrary secret key material values. This results in the calculation of the key wrapping key (KWK) and the random short-lived contribution QeM by the SC. The same KWK is used as an input of the key wrapping function, which wraps the secret key material.

Also, the QeM is returned to the originating party.

- Step 5: The originating party encapsulates the encrypted snapshot, alongside its authentication tag value, the initialization vector, the ephemeral ECMQV contribution QeU , and the wrapped key material. For this step, the platform-independent CMS is used.

- Step 6-9d: The CMS-encapsulated snapshot and the needed data are sent in the form of a MQTT payload via the broker to each of the recipients. Once all of the recipients involved receive the snapshot, they unfold the payload in order to get the public key. This is then used to calculate the key wrapping key which then in turn unwraps the secret key material. Lastly, the recipient decrypts the encrypted snapshot, and authenticates it using the authentication tag.
- Justification for O1: as mentioned above in step 9, the receiving party (the vendor), should verify the integrity and authenticity of the snapshot by unwrapping the wrapped key material. If this step is done successfully, then it can be assumed that it was wrapped by and generated with the claimed SC instance. Moreover, if the decryption phase ends successfully and the snapshots authenticity is verified, one can assume that the unmodified payload really was encrypted by the originating party.

- Justification for O2: As the key agreement scheme requires the private key, only the receiving parties for which the secret key material has been wrapped and included with the encapsulated snapshot are able to perform it in a successful manner. Therefore, the snapshots are protected until an appropriate recipient can successfully perform the key agreement scheme and unwrap the decryption key.

- Justification for O3: All of the encapsulated snapshots are exchanged through the broker using MQTT messages. The message includes some topic information which helps the broker find out the origin of a snapshot. To authenticate the snapshots, the customer is added in the recipient list and can perform the key agreement scheme in a similar way to the vendor.

5.2.3 Message Queue Telemetry Transport

The MQTT Protocol is a lightweight protocol widely used for Internet of Things (IoT)[6] applications. The protocol is used because of its reliability, its simplicity and real time capability. In addition to these features, the protocol is also proved to be secure since it implements the current security standards with regards to encryption and decryption.

Another key feature which led to the decision to use the protocol to fulfill the project requirements were the Quality of Service (QoS)[7] properties of an MQTT message.

The QoS of an MQTT message defines details about the delivery. Since the protocol is used for various purposes there are different types of a message delivery. Delivery types are developed to ensure that a particular client receives exactly a certain number of copies of a message. With respect to the number of deliveries to a recipient, the QoS defines three different standards.

The delivery type with the highest quality of service is the "Exactly once delivery", which does not allow either duplication of messages or loss of messages. The message receiver has to send a reply when the message is received. Due to the MQTT specification[8], the receiver of the message has to perform a two-step acknowledgment to ensure the sender that the message was received. Both parties, the sender as well as the receiver have to perform some actions to aknowledge the receival of a message.

Sender: (Banks and Gupta (2014))

- MUST assign an unused Packet Identifier when it has a new Application Message to publish.
- MUST send a PUBLISH packet containing this Packet Identifier with QoS=2, DUP=0
- MUST treat the PUBLISH packet as "unacknowledged" until it has received the corresponding PUBREC packet from the receiver.

[6]Cisco, 2017(a).
[7]Cisco, 2017(b).
[8]Banks and Gupta, 2014, p. 54.

- MUST send a PUBREL packet when it receives a PUBREC packet from the receiver. This PUBREL packet MUST contain the same Packet Identifier as the original PUBLISH packet.
- MUST treat the PUBREL packet as "unacknowledged" until it has received the corresponding PUBCOMP packet from the receiver.
- MUST NOT re-send the PUBLISH once it has sent the corresponding PUBREL packet

Recipient: (Banks and Gupta (2014))

- MUST respond with a PUBREC containing the Packet Identifier from the incoming PUBLISH Packet, having accepted ownership of the Application Message.
- Until it has received the corresponding PUBREL packet, the Receiver MUST acknowledge any subsequent PUBLISH packet with the same Packet Identifier by sending a PUBREC. It MUST NOT cause duplicate messages to be delivered to any onward recipients in this case.
- MUST respond to a PUBREL packet by sending a PUBCOMP packet containing the same Packet Identifier as the PUBREL.
- After it has sent a PUBCOMP, the receiver MUST treat any subsequent PUBLISH packet that contains that Packet Identifier as being a new publication.

In addition to the reliabilty of the procotol, another significant property of the MQTT is that it is suitable for high-performance devices, such as a server as well as embedded systems like the SmartServiceHub, without losing its key advantages in comparison to other protocols.

Topic subscription and topic hierarchy

The message carries the actual message content, as well as the *topic* (Banks and Gupta, 2014, p. 35), which identifies the information channel on the broker where the message will be posted. The topic is a variable UTF-8 string which must not contain wildcard characters. The wildcard character concept allows to create complex topic hierarchy along with topic filters and topic levels.

The topic hierarchy is made by splitting topic parts with a "/" character. Each string separated by the forward slash represents a topic level. Topic levels can be used by both communication parties, the sender and the receipient. The advantages of the hierarchy can be used for various purposes, some inlude authentication (certificate to topic binding), message parsing and message interpretation.

An example for a topic hierarchy which can be used as a client to a topic binding is "$identifier_1/identifier_2/identifier_3$", where the "$identifier_1$" represent the top level in the hierarchy. Suppose that a client is interested in movies, and let's assume that the value of the top level identifier is "movies". The next level could represent the genre of the movie and could be identified by "thriller". The third level could represent the exact movie name. The whole topic is then represented by "movies/thriller/movie name". Another interesting feature in the topic hierarchy is the wildcard character "#" which can be used anywhere in the topic structure. An example for the usage of the wildcard is replacing a topic level with the wildcard character. If the client is interested in all movies from a specific genre, the previous topic would be used as follows: "movies/thriller/#". Since the usage of the wildcard character is not limited, the user can also replace the second level identifier with the wildcard and retrieve all messages from the first level topic ("movies/#/#" to obtain all messages from the movies topic).

5.3 System components

This section will provide an overview of the components, the used technology and the implementation details.

5.3.1 SmartServiceHub

The *SmartServiceHub* is the component which is installed in a vehicle, with tasks to communicate with the broker, and receiving and installing updates, as well as reading diagnostic data from the vehicle and publishing them to the broker.

As the network traffic is encrypted, this component is also responsible for the decryption of incoming messages (messages from the broker), as well for the encryption of outgoing messages (messages to the broker). In order to perform these operations, the *SmartServiceHub* stores the keys for encryption and decryption. As it is illustrated in the figure 5.3, the *SmartServiceHub* carries its own private key and the backends' public key for encryption of outgoing messages and uses the previously metioned private key for decryption of incoming messages.

In order to get rid of security thretas with regards to open ports, all ports except the MQTT port 8883 are disabled.

Hadware specification

- CPU and operating system
  - CPU: ARM Cortex-A9, 1 GHz (Texas Instruments AM4379)[9]
  - Operating system: Linux (Custom Build Image)[10], kernel version 4.1.6
- Interfaces
  - 2x CAN Interface
  - 1x serial interface
  - 1x LAN
  - 1x USB
  - 1x SD card slot

5.3.2 MQTT Broker

As the usage of an MQTT broker seemed to be promising, which was shown in a research by Priller, Aldrian, and Ebner, 2014, experts continued the verification in (Lesjak, Hein, et al., 2015) and developed the solution which was used by this project to achieve the main project goals.

[9] Texas Instruments, 2017.
[10] *Yocto Project* 2017.

The MQTT broker is a component running on either a windows or a linux machine. The service which implements the MQTT protocol is developed by Eclipse[11] and the current version of the protocol is Eclipse Mosquitto 3.1[12].

In adittion to the main task of the broker which is transporting messages to the endpoint, the MQTT broker also implements security mechanisms proposed in (Lesjak, Hein, et al., 2015, Lesjak, Bock, et al., 2016) to ensure the maximum security level and forbid unauthenticated parties to publish messages.

Client to broker authentication

Each client which is communicating with the broker has to authenticate himself to the broker. The authentication is achieved by using an authenticated TLS channel. In order to be verified by the broker, a client needs a public Broker TLS authentication key (BTAK). The matching private key is securely managed by the broker. The client's public key is also known to the server (MTAK). The corresponding client private key is stored securely by the client.

[11] *Eclipse* 2017.
[12] Eclipse, 2017.

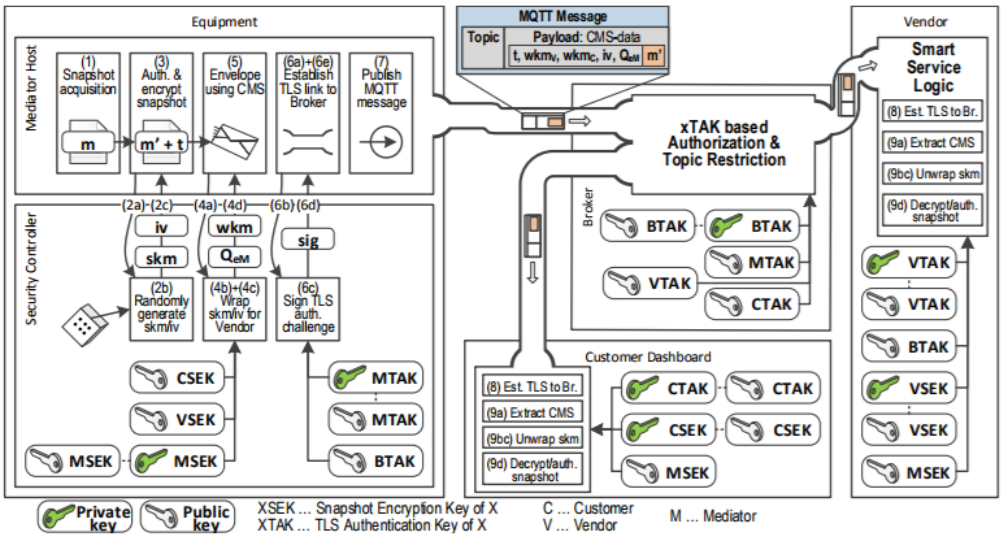


Figure 5.3: MQTT authentication
Source: Lesjak, Bock, et al., 2016

The same authentication procedure is used for backend to broker authentication. The backend, labeled as vendor in the figure 5.3 stores its private key (VTAK) in a secure way, alongside other public keys (BTAK, VTAK).

Certificate to topic authentication scheme

According to the proposed solution in "Securing smart maintenance services: Hardware-security and TLS for MQTT" described in the previous section, the client has to authenticate himself to the broker in order to publish a message. The autentifiation is implemented by the certificate to topic binding. A client can publish a message on a certain topic, only if the client is in possesion of a certificate which belongs to the given topic.

5.3.3 Backend

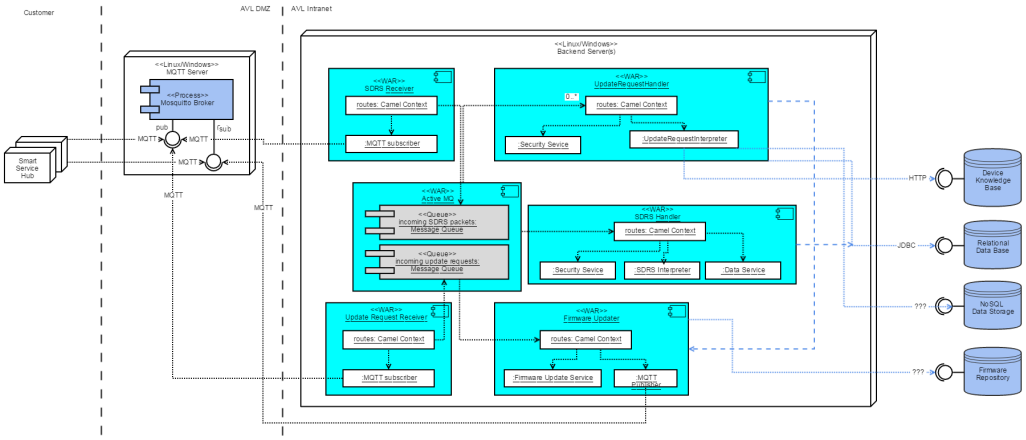


Figure 5.4: Backend Component Diagram

DMZ protection

In order to protect the backend components which include the network, the application and the host, a demilitarized zone is used. DMZs are a common approach for secure hosting and securing the enterprise network. In general, the DMZ acts as the security barrier between the external (unsafe) and the internal (safe) network. As the DMZ operates between two networks, the local network and external the DMZ is guarded by two firewalls between the networks. According to Stouffer, Falco, and Scarfone, 2013, DMZ usually blocks or controls the unnecessary traffic and protocols (SNMP[13], ICMP[14], RPC[15], FTP[16]). Furthermore, such a network topology is ideal to secure confidential data. As shown in 5.1, the DMZ is guarding the MQTT broker by allowing only incoming messages on the 8883 port (Banks and Gupta, 2014, Chapter 4.2).

[13]Case et al., 1990.
[14]J. Postel, 1981.
[15]Thurlow, 2009.
[16]J. Postel, 1980b.

LAN protection

Additionally to the DMZ, sensitive data (ECU update files, key material, vehicle data, diagnostic data) is also protected by a Local Area Network (LAN)[17]. The main purposes of the LAN are to provide data integrity and to block all the outgoing traffic, except the messages to the port 8883, since the communication between the backend from the LAN is realized over the previously mentioned MQTT port, as it is illustrated in the figure 5.1.

Backend components

In favor of the fine granularity of the overall system, the backend functionality is divided into various services with their corresponding tasks. In addition to the fine granularity, such architecture is also beneficial with regards to the system maintainability. As the project deals with two major scenarios, the components are classified into two categories, the first one, components which are reposponsible for the ECU update scenario, and the second one, components which are responsible for collecting and triggering telemetry data exchange.

- ECU update components
  - *UpdateRequestReceiver*
  - *UpdateRequestHandler*
  - *UpdateRequestInterpreter*
- Telemetry data processing
  - Collection of telemetry data
    - *SDRSReceiver*
    - *SDRSHandler*
    - *SDRSInterpreter*
  - Request for telemetry data
    - *DBCFilePublisher*

[17] Velde et al., 2007.

UpdateRequestReceiver

The task of the *UpdateRequestReceiver* is, as the name aleady reveals, the collection of update requests from the broker. As described in previous chapters, the data from the SmartServiceHub is published on the broker, since the broker implements complex hierarchy in regards to the message topics, the *UpdateRequestReceiver* distinguishes the origin of the message by inspecting the topic on which the message was published.

The update request processing starts with the decryption of the content received from the broker, the *UpdateRequestReceiver* queries the database in order to find the client certificate which corresponds to the topic. (see 5.3.2). The whole procedure will be explained in the 6 chapter.

UpdateRequestHandler

This component is responsible for the update request handling. The task of the *UpdateRequestHandler* is to forward the data which is previously received and decrypted by the *UpdateRequestReceiver* to the next component in the chain, namely the *UpdateRequestInterpreter*.

The simple design of this component allows performing changes to the request interpretation in case that the requirements change over time.

UpdateRequestInterpreter

The *UpdateRequestInterpreter* component is the last component in the update request processing chain, the task of this component is to parse the data from the request, and store it accordingly into the database. Correspoding to the message content, the interpreter queries the database in order to determine the current ECU software versions of the vehicle which sent the request for the update.

According to the software versions which are retrieved from the database for a specific vehicle, the interpreter can now generate the update package

and send it back to the vehicle. This procedure will be explained in the implementation chapter in full detail.

SDRSReceiver

The *SDRSReceiver* (Service Data Record Set Receiver) is the component responsible for receiving messages for the telemetry data use case. The idea behind this receiver component is the same as by the previous receiver component the *UpdateRequestReceiver*. As the idea remains the same, the benefits of the design also remain the same, the architecture allows various modifications which is an important factor with regards to the maintainability and the reusage of the component.

SDRSHandler

The component responsible for forwarding messages from the *SDRSReceiver* to the *SDRSInterpreter*. Since the component acts like dispatcher of messages, it is possible to add other features before, or after the message forwarding in order to, for example, analyze the data, monitor the traffic and so on.

SDRSInterpreter

After receiving the data from the *SDRSHandler*, the *SDRSInterpreter* has the task to extract the information which is sent in the Comma-Separated Values (CSV)[18] format. After extracting the data, the interpreter then stores it accordingly into the database.

DBCFilePublisher

Is a standalone component which has the responsibility to send DBC[19] files to the broker, in order to read diagnostic/telemetry data from a specific

[18]Shafranovich, 2005.
[19]Vector Informatik GmbH, 2007.

vehicle. The component publishes DBC files to the broker periodically. The idea behind the continuous publishing of DBC files is that the backend does not know when the vehicle is able to receive the file since there is no indication when a vehicle has a network access. The sending frequency is variable, one can define the sending frequency arbitrarily.

Database

Alongside components which are responsible for data interpretation, and receiving data, the main component which is used for storing the data is the Oracle database which is commonly used for high-performance database applications[20].

The database carries various information which is sent and received by the backend in order to accomplish the update and the telemetry data scenario. In addition to the data which is sent across the network, the database also stores meta information about vehicles, ECUs, ECU versions, vehicle manufacturers, as well as data to observe and analyze the vehicle network, the ones include, storage of update requests, update install logs, etc.

Another task for which the database is responsible for, is the storage of firmware update packages. Furthermore, there are two possible ways to store update packages, one can store an update package with, or without a digital signature. The case where the signature of the update is stored along the actual update package is guaranteeing a higher security level since the backend can prove the origin of the update based on the digital signature. The idea behind these two ways of storing data is to support both, car manufacturer which are delivering updates without a digital signature and the ones which are signing update packages before the delivery.

In addition to features regarding the data traffic storage, the database is also storing the key-material which is needed for encryption and decryption of the content which is sent across the network, as well as the key material which is needed for the backend to MQTT broker authentication. Furhermore, the database manages the private key which is used for the encryption of all messages which are sent to the broker, including the update packages

[20]Burleson, 1996.

and the DBC files. As described in the section 5.2.2, the data is encrypted for a specific recipient, since such an encryption requires the recipients' public key, public key of receipients (vehicles) are also stored in the database.

5.4 Countermeasures against threats

According to the analysis in the chapter 3, the project should be able to cover threats like, spoofing, network sniffing, the cryptography issues and so on. The table below illustrates the common threats, countermeasures to avoid the threats as well as the solutions provided by this projects with the reference how a specific threat is mitigated.

Threat	Countermeasures	Achieved by	Chapter
Spoofing user identity	Strong authentication Credentials in plaintext	TLS Broker authentication	5.2.1, 5.3.2
Tampering	Data hashing and signing. Digital signatures. Strong authorization. Provide message integrity.	CMS encryption Data Signing Broker authorization	5.2.2, 5.3.2
Repudiation	Use digital signatures.	Data signing	5.2.2
Information disclosure	Strong authorization. Strong encryption. Ensure message confidentiality.	CMS encryption TLS	5.2.2, 5.2.1
Denial of service	Use resource and bandwidth throttling techniques. Validate and filter input.	Broker authorization DMZ	
Information gathering	Firewall Disable unused ports	DMZ Firewall	5.3.2, 5.3.3
Sniffing	Strong physical security Strong encryption	DMZ Firewalls TLS CMS encryption	5.3.3, 5.2.1, 5.2.2
Session hijacking	Encrypted session negotiation Strong encryption	TLS CMS encryption Broker authorization	5.2.1, 5.2.2, 5.3.2
Input validation	Prepared statements (SQL) Proper input validation	Broker authorization Java prepared statements	5.3.2 , 5.3.3
Sensitive data	Transffering over TLS Signing data Strong encryption	TLS CMS encryption	5.2.1, 5.2.2
Cryptography	Strong encryption Key expiration	CMS encryption Key freshness	5.2.2

Table 5.2: Security requierements and solutions

6 Implementation

As described in the previous chapters, the project deals with two major scenarios. One possible use case is the ECU update over the air and the other is the exchange of telemetry as well as the diagnostic data over the air. Both use cases use the same architecture as described in the chapter 5.

6.1 Update Scenarios

The use case for the over the air update of ECUs will be explained in the following sections. The process of the update request and update delivery is illustrated in the figure 6.1.

6.1.1 Update Request and Update Response Process

The update process starts with the process initiator, the vehicle sends an update request message to the broker. The main reason why the initiator role is assigned to vehicles is due to the fact that is hard to predict when a specific vehicle is online and able to exchange data with the backend.

After the request is prepared and sent, the initiator subscribes to the update topic and waits for the response from the backend. In case that the vehicle is not up-to-date, updates are available and the vehicle is supposed to install these updates and send the update install log back to the backend. The log contains information on the installation process as well as the information whether the installation was successful or not. The log is then stored in the database and the update process is done. The whole process is explained in more detail in the following sections.

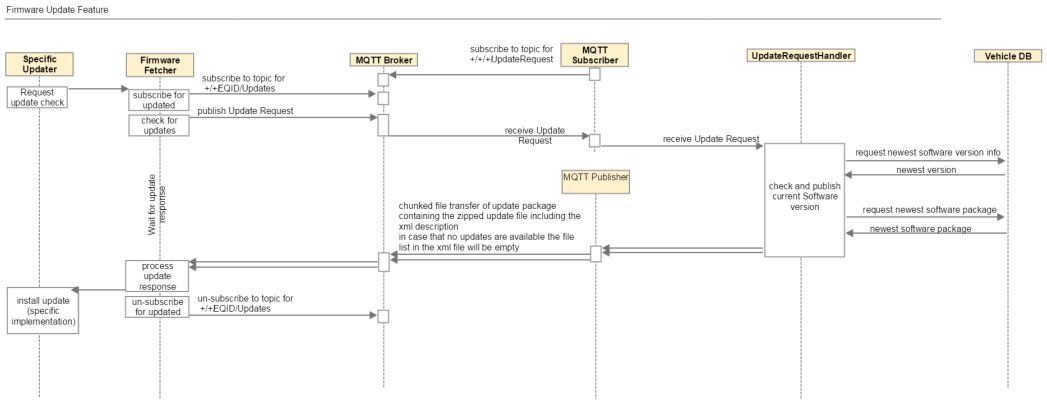


Figure 6.1: Update Request Diagram

6.1.2 Update Request

In order to keep track of software versions in vehicles a database is used. The database holds infomation about vehicles, their ECUs as well as software versions of a particular ECU. In addition to vehicle and ECU data, the database also carries data about old versions and the history of updates and update requests.

In case that a vehicle is online, the *SmartServiceHub* sends an update request message to the broker in form of a Extensible Markup Language (XML)[1] file. The XML Format is described later in the text. Based on the authentication scheme which is described in the previous chapter, every vehicle has its own update request topic on the broker. According to the scheme, the topic is defined as ”Vehicle Producer/Vehicle Identification Number/updaterequests”. The first part of the topic describes the vehicle producer, for example: ”Mercedes”, ”Audi”, ”JAC”. The second part is the unique *Vehicle Identification Number*, often called “chassis number”. The last part is a fixed string ”updaterequests” which is common for all update requests of all vehicles.

[1] Rosenberg, 2007.

Update Request format

```
<?xml version="1.0" encoding="UTF-8"?>
<update_request>
   <timestamp>LINUX-TS</timestamp>
   <devices>
      <device id="...">version</device>
      <device id="...">version</device>
   </devices>
</update_request>
```

In order to keep track of update requests, the XML string carries information on the time in which the request was created. The format of the timestamp is the same as in all UNIX operating systems, which count time in milliseconds since 1.1.1970[2].

The next node is a parent node for all devices which are requesting an update. "Device id" carries the control unit identification number. Every update request is logged into the vehicle database.

Include dashboard image with the update requests

Update Request Processing

As described above, the request is sent as an XML file. This file is encrypted using a client private key. When the message is received by the broker, a microservice for decryption is executed. The decrypted message is forwarded to another service which has the responsibility to interpret the data. The parsed values, request timestamp, and device identification numbers are retrieved. The request timestamp is then stored in the database along with the corresponding device identification numbers which are requesting an update.

After the data logging, the actual update request is performed in the backbone of the system. The database is queried with the information obtained from the vehicle control unit ids. Since all the data is stored in the database,

[2]UNIX, 2017.

no interaction with the vehicle is needed. Queries are made to the database for a new update for every single device id from the update request. If the current software version for a given control unit id is outdated, a new version will be prepared for the delivery.

6.1.3 Update Response

In both cases, whether updates are available or not, a response package will be sent. The update package consists of an XML description file and update file(s). All files are compressed in an archive to reduce the transmission costs. The encrypted and BASE64 encoded update packages are then ready for delivery. There are two different cases for the update response.

Based on the information retrieved in the process described above, an update response message is generated. In both cases, whether an update exists or not an update response message will still be generated and sent.

Updates Available

In the more interesting case, updates are available.

Depending on the number of updates available, an XML file is produced. The file carries meta information about the update files. An example XML file could look like the following:

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

The XML has a timestamp so one can keep track of the time when a certain update for a specific vehicle was sent. Another interesting feature of the description file is the file order attribute. In case that updates depend on each other, the file order attribute define the order in which the updates have to be installed.

Other data which can be found in the XML file include the name of the update file, the file format and the file size in a certain unit which is listed as an attribute in the same XML node.

In order to verify the update file origin, an optional signature file can also be part of the update package. In the XML file, only the signature file name is added as an attribute. However, the file is stored in the archive, and the SmartServiceHub has to verify the update file and the appended signature. If the signature matches to the origin, the update is valid, otherwise the update will be treated as malicious and will be discarded.

The last attribute is the unique device id of the device which needs to be updated.

Update Install Log

After the vehicle receives the update files, the installation of updates is executed. During this time, the SmartServiceHub logs the installation process into a file. The log file contains detailed information about the installation progress. The log also contains information concerning the memory blocks

which have been modified, the UDS Protocol status messages and the final status message from the update installation. After the installation is finished, the log file containing the debug output from the vehicle is sent back to the backend.

A service on the backend side is responsible for the interpretation of the log. The service is able to distinguish whether the installation was successful or not. In case that the installation succeeded, the software versions for the updated control units are updated.

No Updates Available

In case that no updates are available, a similar XML file will be sent back to the vehicle. However, since all vehicle control units are up-to-date, the number of update files will be 0. Sending a message even when no updates are available was a design decision with the advantage that the procedure stays the same. Another key argument for sending messages even when no updates are available, is the fact that the vehicle does not know when the procedure is over and the system with a response to every request provides an appropriate solution.

The XML response, in this case, carries only the information about the current timestamp when the response was generated.

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

6.2 Telemetry Data

The other use case of the project was to develop a scenario of collecting diagnostic and telemetry data in a secure way over the air. The communication protocols remain the same, the main difference is that the initiator role

is assigned to the to the backend. Since the backend does not know when a vehicle is online and able to exchange telemetry data with the backbone of the system, the request for the exchange of telemetry data is triggered and sent from the backend periodically. The request is defined in a similar way as an update response package. There are two files compressed in an archive which is sent to every vehicle in a defined period.

6.2.1 Telemetry Data Request Package

XML Telemetry Description File

```
<?xml version="1.0" encoding="UTF-8"?>
<telemetry_data>
   <frequency>60000</frequency>
   <duration>10000</duration>
</telemetry_data>
```

As it is described in the XML file, there are two important parameters which define the sending frequency in milliseconds and also the duration of the capture of specific signals and CAN messages. These two parameters have the advantage that the backend can periodically request and monitor specific signals in order to inspect possible sources of errors.

DBC File

The XML file described above is sent with the correspoding DBC file. The DBC (DBC Communication Database for CAN) file describes the signals which the backend wants to capture.

```
VERSION "HY_CAN"


NS_ :
NS_DESC_
```

```
CM_
BA_DEF_
BA_
VAL_
CAT_DEF_
CAT_
FILTER
BA_DEF_DEF_
EV_DATA_
ENVVAR_DATA_
SGTYPE_
SGTYPE_VAL_
BA_DEF_SGTYPE_
BA_SGTYPE_
SIG_TYPE_REF_
VAL_TABLE_
SIG_GROUP_
SIG_VALTYPE_
SIGTYPE_VALTYPE_
BO_TX_BU_
BA_DEF_REL_
BA_REL_
BA_DEF_DEF_REL_
BU_SG_REL_
BU_EV_REL_
BU_BO_REL_
SG_MUL_VAL_

BS_:

BU_: EMS HVCU
VAL_TABLE_ TMDirCmd 2
"rotating backward" 1 "rotating forward" 0 "not rotating" ;
VAL_TABLE_ DCCOperMode 2
"reserved" 1 "Buck Mode" 0 "Standby" ;
```

```
BO_ 265 HVCU_FrP08: 8 HVCU
 SG_ HVCU_Cnt109 : 15|4@0+ (1,0) [0|15] ""  EMS

BO_ 267 HVCU_FrP09: 8 HVCU
SG_ HVCU_Cnt10B : 15|4@0+ (1,0) [0|15] ""  EMS
```

Since signals are captured directly from the CAN bus, one can adapt the DBC file and capture virtually everything on the CAN bus. Furhermore, by the project definition, DBC files are sent periodically to vehicles, that gives the administrators the opportunity to change the capture signals in order to get different data from the vehicle.

7 Conclusion

The scope of this thesis was the research of the global security state within the automotive industry as well as the research on common shortcomings with regards to backend security. The requirements for this project were defined after the classification of the flaws which are summarized in chapters 3.4 and 4.3 the main ones with regards to vehicle security include, CAN message modifications, custom CAN messages injection, problems caused by open ports. The typical threats regarding the backend security include, sniffing, spoofing, weak encryption, unsafe protocols, wrong host and network configurations and other.

In addition to the researches on security, this project also focuses on finding an elegant solution to one of two scenarios in the backend, the update of ECUs over the air, as well as the gathering of diagnostic data over the air. Furthermore, the project also provides a concept for data management, key management and visualization of the information collected from a vehicle. Alongside the main objective which was the system security, other objectives have been covered as well. An important requirement, the usability, is covered by the way of the update transmission and installation. The solution found is a rather large improvement in comparison to old ECU update methods which require the physical presence of a vehicle in, for example, a vehicle shop.

From the technical point of view, an important aspect of this project is the capability of it to be reused for other purposes with similar requirements, especially with regards to the secure communication, the architecture, as well as the project design. Another argument in favor of the reusage of technology is the fact that this project inherited basic principles and concepts from the Arrowhead[1] project. Oher important requirements are reached by

[1] Arrowhead, 2017(b).

the easy maintainability of the project. As already described in the chapter 4.1.3, the costs for the system maintenance are in the range from 65% to 75% and therefore an important requirement was to develop an easily maintainable system to reduce the maintenance costs.

Although this project tried to focus on the most frequent known flaws which were detected in recent years, there is plenty of room to improve in the modern automotive industry.

Topics which could potentially be covered by this project as a base will be explained in the next section.

8 Outlook and future work

Although the system proved to be reliable and showed no weaknesses, goals regarding future improvements have already been set. The main focus will be moved to secure vehicle-to-vehicle communication. It is planned for vehicles to be able to communicate and share data with each other, in the same manner in which the communication has been established between the backend and frontend. Another focal point could be the payment of toll roads though the implemented communication.

With regards to the backend, an improvement could be reached by implementing a hierarchy permission model for update distribution. Such a model would be beneficial with regards to the updates of vehicle groups. That role model would allow classifying vehicles into different categories with regards to the vehicle's equipment, the vehicle manufacturer, the vehicle model and the vehicle build year. Furthermore, one could launch update only for vehicles from a certain group, for example, one could update all vehicles built in a certain year with a specific equipment package.

As the main focus was the security of the system, an important aspect which is not covered, but has to be covered in the future is the functional safety. As defined in Brunel et al., 2016, the functional safety is the part of the overall safety which depends on a system operating correctly in response to its inputs. Aspects with regards to functional safety within the project include the installation of updates when the vehicle is in a certain state. In order to cover the mentioned aspect, one has to make sure that the vehicle is able to install the update, the update installation ready-state can be achieved by ensuring that the vehicle is not in motion, other preconditions include the availability of memory resources in case of large updates as well as a trustworthy network connection which guarantees a reliable data exchange.

Since the project i capable to securely distribute updates for ECUs over the air, another approach to extend the project functionality could be to tranfer updates for the *SmartServiceHub* in order to perform an firmware update of the *SmartServiceHub*.

One possible approach for the future could be to support real time data exchange in order to supply the backend continuously with data from the vehicle. The main issue with regards to the real time data exchange is due to the vehicle connectivity which can be interrupted.

The most important asset for the future will be the fallback functions in case that the update process fails. However, the design and the implementation of these functions is rather complex. A possible approach to achieve such functionality would be to save the current firmware update on the *SmartServiceHub* before installing the newest software and install the software again if the installation of the newer software fails. Another approach could be to implement the logic for such scenario on the backend. In case that the update fails, the backend can send the older software in order to get rid of the problem caused with the newer software.

Bibliography

Arrowhead (2017[a]). *Arrowhead Framework*. URL: https://forge.soa4d.org/plugins/mediawiki/wiki/arrowhead-f/index.php/Main_Page (visited on 08/30/2017) (cit. on p. 1).

Arrowhead (2017[b]). *Arrowhead Project*. URL: http://www.arrowhead.eu/ (visited on 06/05/2017) (cit. on pp. 1, 43, 73).

Automotive Techis (2017). *CAN Bus*. URL: https://automotivetechis.wordpress.com/2012/06/01/ (visited on 08/13/2017) (cit. on p. 6).

Banks, A. and R. Gupta (2014). *MQTT version 3.1.1*. URL: http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/csprd02/mqtt-v3.1.1-csprd02.pdf (cit. on pp. 51, 52, 57).

Bellovin, S. and R. Housley (2005). *Guidelines for Cryptographic Key Management*. BCP 107. RFC Editor (cit. on p. 31).

Berners-Lee, Tim, Larry Masinter, and Mark McCahill (1994). *Uniform Resource Locators (URL)*. RFC 1738. http://www.rfc-editor.org/rfc/rfc1738.txt. RFC Editor. URL: http://www.rfc-editor.org/rfc/rfc1738.txt (cit. on p. 38).

Blake-Wilson, S., D. Brown, and P. Lambert (2002). *Use of Elliptic Curve Cryptography (ECC) Algorithms in Cryptographic Message Syntax (CMS)*. RFC 3278. RFC Editor (cit. on p. 46).

BMW (2017). *ConnectedDrive*. URL: https://www.bmw.com/en/topics/offers-and-services/bmw-connecteddrive-services/mobile-devices.html#firmware.html (visited on 09/03/2017) (cit. on p. 9).

Bos, Joppe W. et al. (2014). "Elliptic Curve Cryptography in Practice." In: *Financial Cryptography and Data Security*. Springer Berlin Heidelberg, pp. 157–175. DOI: 10.1007/978-3-662-45472-5_11. URL: https://doi.org/10.1007/978-3-662-45472-5_11 (cit. on p. 29).

Brunel, Julien et al. (2016). *Recommendations for Security and Safety Co-engineering (Release no3) - Part B*. en. DOI: 10.13140/rg.2.1.3649.1923. URL: http://rgdoi.net/10.13140/RG.2.1.3649.1923 (cit. on p. 75).

Burleson, Donald K. (1996). *High-performance Oracle database applications*. pub-CORIOLIS:adr: Coriolis Group Books, pp. xiii + 461. ISBN: 1-57610-100-2 (cit. on p. 61).

Case, Jeffrey D. et al. (1990). *Simple Network Management Protocol (SNMP)*. STD 15. http://www.rfc-editor.org/rfc/rfc1157.txt. RFC Editor. URL: http://www.rfc-editor.org/rfc/rfc1157.txt (cit. on p. 57).

Checkoway, Stephen et al. (2011). *Comprehensive Experimental Analyses of Automotive Attack Surfaces*. URL: http://www.autosec.org/pubs/cars-usenixsec2011.pdf (cit. on p. 14).

Cheminod, Manuel, Luca Durante, and Adriano Valenzano (2013). "Review of Security Issues in Industrial Networks." In: *IEEE Transactions on Industrial Informatics* 9.1, pp. 277–293. DOI: 10.1109/tii.2012.2198666. URL: https://doi.org/10.1109/tii.2012.2198666 (cit. on p. 31).

Cisco (2017[a]). *IoT*. URL: http://www.cisco.com/c/dam/en_us/solutions/trends/iot/introduction_to_IoT_november.pdf (visited on 05/15/2017) (cit. on p. 51).

Cisco (2017[b]). *QoS*. URL: http://www.cisco.com/c/en/us/td/docs/ios/12_2/qos/configuration/guide/fqos_c/qcfintro.pdf (visited on 07/30/2017) (cit. on p. 51).

Dhanjani, Nitesh (2014). URL: http://www.dhanjani.com/blog/2014/03/curosry-evaluation-of-the-tesla-model-s-we-cant-protect-our-cars-like-we-protect-our-workstations.html (cit. on p. 17).

Dierks, T. and E. Rescorla (2008). *The Transport Layer Security (TLS) Protocol Version 1.2*. RFC 5246. http://www.rfc-editor.org/rfc/rfc5246.txt. RFC Editor. URL: http://www.rfc-editor.org/rfc/rfc5246.txt (cit. on p. 44).

Eclipse (2017). URL: http://www.eclipse.org/ (visited on 04/03/2017) (cit. on p. 55).

Eclipse (2017). *Eclipse Mosquitto*. URL: https://projects.eclipse.org/projects/technology.mosquitto (visited on 08/07/2017) (cit. on p. 55).

Eddy, W. (2007). *TCP SYN Flooding Attacks and Common Mitigations*. RFC 4987. RFC Editor (cit. on p. 37).

Fielding, Roy T. et al. (1999). *Hypertext Transfer Protocol – HTTP/1.1*. RFC 2616. http://www.rfc-editor.org/rfc/rfc2616.txt. RFC Editor. URL: http://www.rfc-editor.org/rfc/rfc2616.txt (cit. on p. 40).

Foster, Ian et al. (2015). *Fast and Vulnerable: A Story of Telematic Failures*. URL: https://www.usenix.org/system/files/conference/woot15/woot15-paper-foster.pdf (cit. on pp. 18, 20, 21).

GlobalCarsBrands (2017). *Top 10 Newest Car Technologies That Have Revolutionized the Auto Industry*. URL: https://www.globalcarsbrands.com/top-10-newest-car-technologies-that-have-revolutionized-the-auto-industry/ (visited on 05/08/2017) (cit. on p. 8).

Horman, Simon (2005). *SSL and TLS An Overview of A Secure Communications Protocol*. URL: http://horms.net/projects/ssl_and_tls/stuff/ssl_and_tls.pdf (cit. on p. 45).

Housley, R. (2009). *Cryptographic Message Syntax (CMS)*. STD 70. http://www.rfc-editor.org/rfc/rfc5652.txt. RFC Editor. URL: http://www.rfc-editor.org/rfc/rfc5652.txt (cit. on p. 45).

Management of information and communications technology security – Part 1: Concepts and models for information and communications technology security management (2004). ISO. Geneva, CH: International Organization for Standardization (cit. on p. 41).

Johansson, Karl Henrik, Martin Törngren, and Lars Nielsen (n.d.). *Vehicle Applications of Controller Area Network*. Tech. rep. URL: http://people.kth.se/~kallej/papers/can_necs_handbook05.pdf (cit. on p. 7).

Krawczyk, Hugo, Mihir Bellare, and Ran Canetti (1997). *HMAC: Keyed-Hashing for Message Authentication*. RFC 2104. http://www.rfc-editor.org/rfc/rfc2104.txt. RFC Editor. URL: http://www.rfc-editor.org/rfc/rfc2104.txt (cit. on p. 40).

Kumar, Balraj (2012). "A Survey of Key Factors Affecting Software Maintainability." In: *2012 International Conference on Computing Sciences*. IEEE. DOI: 10.1109/iccs.2012.5. URL: https://doi.org/10.1109/iccs.2012.5 (cit. on p. 26).

Lesjak, Christian, Holger Bock, et al. (2016). "Hardware-secured and transparent multi-stakeholder data exchange for industrial IoT." In: *2016 IEEE 14th International Conference on Industrial Informatics (INDIN)*. IEEE. DOI: 10.1109/indin.2016.7819251. URL: https://doi.org/10.1109/indin.2016.7819251 (cit. on pp. 47–49, 55, 56).

Lesjak, Christian, Daniel Hein, et al. (2015). "Securing smart maintenance services: Hardware-security and TLS for MQTT." In: *2015 IEEE 13th International Conference on Industrial Informatics (INDIN)*. IEEE. DOI: 10.

1109/indin.2015.7281913. URL: https://doi.org/10.1109/indin.2015.7281913 (cit. on pp. 54–56).

Mazur, David C. et al. (2016). "Defining the Industrial Demilitarized Zone and Its Benefits for Mining Applications." In: *IEEE Transactions on Industry Applications* 52.3, pp. 2731–2736. DOI: 10.1109/tia.2016.2530045. URL: https://doi.org/10.1109/tia.2016.2530045 (cit. on p. 31).

Meier, J.D. et al. (2003). *Improving Web Application Security Threats and Countermeasures*. URL: https://msdn.microsoft.com/en-us/library/ff649874.aspx (visited on 08/10/2017) (cit. on pp. 35, 38).

Menezes, Alfred John, Paul C. van Oorschot, and Scott A. Vanstone (1996). *Handbook of Applied Cryptography*. Taylor & Francis Inc. 810 pp. ISBN: 9780849385230. URL: http://www.ebook.de/de/product/3648186/alfred_john_menezes_paul_c_van_oorschot_scott_a_vanstone_handbook_of_applied_cryptography.html (cit. on pp. 32–34).

Microsoft (2017). *Microsoft*. URL: https://www.microsoft.com/en-us/ (visited on 08/08/2017) (cit. on p. 35).

Microsoft STRIDE (2017). *The STRIDE Threat Model*. URL: https://msdn.microsoft.com/en-us/library/ee823878(v=cs.20).aspx (visited on 08/07/2017) (cit. on p. 34).

Miller, Charlie and Chris Valasek (2014). *Adventures in Automotive Networks and Control Units*. URL: https://www.ioactive.com/pdfs/IOActive_Adventures_in_Automotive_Networks_and_Control_Units.pdf (cit. on p. 16).

Miller, Charlie and Chris Valasek (2015). *Remote Exploitation of an Unaltered Passenger Vehicle*. URL: http://illmatics.com/Remote%20Car%20Hacking.pdf (cit. on pp. 8, 11, 14–16).

Motorsportzenter (2017). URL: http://www.motorsportscenter.com/printer_26.shtml (visited on 07/03/2017) (cit. on p. 7).

Postel, J. (1980a). *DoD standard Internet Protocol*. RFC 760. RFC Editor (cit. on p. 34).

Postel, J. (1980b). *File Transfer Protocol specification*. RFC 765. RFC Editor (cit. on p. 57).

Postel, J. (1981). *Internet Control Message Protocol*. STD 5. http://www.rfc-editor.org/rfc/rfc792.txt. RFC Editor. URL: http://www.rfc-editor.org/rfc/rfc792.txt (cit. on p. 57).

Postel, Jon (1981). *Internet Protocol*. STD 5. http://www.rfc-editor.org/rfc/rfc791.txt. RFC Editor. URL: http://www.rfc-editor.org/rfc/rfc791.txt (cit. on p. 37).

Priller, Peter, Andreas Aldrian, and Thomas Ebner (2014). "Case study: From legacy to connectivity migrating industrial devices into the world of smart services." In: *Proceedings of the 2014 IEEE Emerging Technology and Factory Automation (ETFA)*. IEEE. DOI: 10.1109/etfa.2014.7005136. URL: https://doi.org/10.1109/etfa.2014.7005136 (cit. on p. 54).

Rosenberg, J. (2007). *Extensible Markup Language (XML) Formats for Representing Resource Lists*. RFC 4826. RFC Editor (cit. on p. 64).

Shafranovich, Y. (2005). *Common Format and MIME Type for Comma-Separated Values (CSV) Files*. RFC 4180. http://www.rfc-editor.org/rfc/rfc4180.txt. RFC Editor. URL: http://www.rfc-editor.org/rfc/rfc4180.txt (cit. on p. 60).

S. Muthanna, K. Kontogiannis and K. Ponnambalam, eds. (2000). *A maintainability model for industrial software systems using design level metrics*. (Nov. 25, 2000). Ieee. ISBN: 0-7695-0881-2. DOI: 10.1109/WCRE.2000.891476. URL: https://www.amazon.com/Seventh-Working-Conference-Reverse-Engineering/dp/0769508812?SubscriptionId=0JYN1NVW651KCA56C102&tag=techkie-20&linkCode=xm2&camp=2025&creative=165953&creativeASIN=0769508812 (cit. on p. 26).

Stouffer, Keith, Joe Falco, and Karen Scarfone (2013). *Guide to Industrial Control Systems (ICS) Security : Supervisory Control and Data Acquisition (SCADA) Systems, Distributed Control Systems (DCS), and Other Control System Configurations such as Programmable Logic Controllers (PLC)*. Tech. rep. DOI: 10.6028/nist.sp.800-82r1. URL: https://doi.org/10.6028/nist.sp.800-82r1 (cit. on p. 57).

Tesla (2017). *Software Updates*. URL: https://www.tesla.com/support/software-updates (visited on 08/07/2017) (cit. on p. 9).

Texas Instruments (2017). *ARM Cortex-A9*. URL: http://www.ti.com/product/AM4379 (visited on 08/07/2017) (cit. on p. 54).

Thurlow, R. (2009). *RPC: Remote Procedure Call Protocol Specification Version 2*. RFC 5531. http://www.rfc-editor.org/rfc/rfc5531.txt. RFC Editor. URL: http://www.rfc-editor.org/rfc/rfc5531.txt (cit. on p. 57).

UConnect (2017). URL: http://www.driveuconnect.com/ (visited on 08/01/2017) (cit. on p. 16).

UNIX (2017). *UNIX Timestamp*. Ed. by UNIX. URL: http://pubs.opengroup.org/onlinepubs/9699919799/basedefs/V1_chap04.html#tag_04_16 (visited on 07/18/2017) (cit. on p. 65).

Vector Informatik GmbH (2007). *DBC File Format Documentation*. URL: https://wenku.baidu.com/view/41c36d25ed630b1c59eeb534.html## (cit. on p. 60).

Velde, G. Van de et al. (2007). *Local Network Protection for IPv6*. RFC 4864. RFC Editor (cit. on p. 58).

Xing, Wang, Huiyan Chen, and Huarong Ding (n.d.). "The application of controller area network on vehicle." In: *Proceedings of the IEEE International Vehicle Electronics Conference (IVEC'99) (Cat. No.99EX257)*. IEEE. DOI: 10.1109/ivec.1999.830728. URL: https://doi.org/10.1109/ivec.1999.830728 (cit. on p. 7).

Yocto Project (2017). URL: https://www.yoctoproject.org (visited on 08/09/2017) (cit. on p. 54).