Dipl.-Ing. Marco Steger, BSc

Secure and Efficient Wireless Automotive Software Updates

DISSERTATION

to achieve the university degree of

Doktor der Technischen Wissenschaften

submitted to

Graz University of Technology

Supervisor

Univ.-Prof. Dipl.-Inform. Dr. sc. ETH Kay Uwe Römer

Institute of Technical Informatics

Advisor

Ass.Prof. Dott. Dott. mag. Dr.techn. MSc Carlo Alberto Boano

Institute of Technical Informatics

Graz, April 2018

Acknowledgements

Writing this thesis was one of the most valuable and rewarding things I did in my life so far. I got to see different places around the globe, was allowed to meet a bunch of great people and attend interesting conferences and other exciting events, and even found a second home in Sydney, Australia. However, there have also been frustrating moments and times where I doubted myself for even starting the whole mission. Especially in these hard times, I could always count on the great support of my professor, my colleagues, my friends, and, of course, my wonderful girlfriend helping me to stay focused and to keep on track along the entire journey. Finally, it is time to thank all of you for your great and never-ending support. I will be thankful for the rest of my life!

I begin by thanking my supervisor, Prof. Kay Römer, for being a great mentor and for guiding me through the last four years. You not only helped me to significantly improve my technical skills and my writing ability, but also provided me with a clear way towards finalizing this thesis and even beyond. Furthermore, you immediately supported the idea of spending some months abroad to enrich my thesis with inputs from other researchers and acted as one of the main enablers of my stay at UNSW in Sydney. Thank you very much for believing in me the whole time, for providing valuable feedback – often even hours before the deadline – on our papers, and for helping me writing this thesis.

I also want to thank Prof. Salil Kanhere for allowing me to come to UNSW, for providing me with a great environment in Sydney to continue my work, and especially for integrating me in your incredible team from the first day on. I'm glad that we were able to continue our common work even once I left Australia. I also want to thank you for agreeing to be the second examiner of my thesis! Additionally, I want to thank my Australian colleagues and friends. Especially Arash Shaghaghi for being a great colleague and friend, and for all the great technical (and non-technical) discussions we had during my stay. Also I want to thank Ali Dorri for introducing me to the Blockchain technology and for always being open for new ideas and research areas. I hope we continue our fruitful collaboration for a long time.

A special thanks also goes to my colleagues from Virtual Vehicle. I want to thank Prof. Daniel Watzenig for allowing me to start my scientific career at Virtual Vehicle and especially for motivating me to start working on my thesis in the first place. Furthermore, thank you so much for supporting my stay in Australia. I doubt that I would have been able to gain this great experience without your advice and support. I also want to thank Michael Karner for the countless discussions about my thesis and its organizational as well as technical aspects (especially during our valued lunch breaks). You were one of the key motivators for me to actually start working on this thesis. A special thanks also goes to Joachim Hillebrand and Werner Rom for supporting me with their technical expertise. Your valuable feedback on our papers really helped to push them to the next level.

I especially want to thank Carlo Alberto Boano for supporting me in some many different ways. Working with you was a great experience and I'm deeply thankful for every feedback on our papers, for every new idea we discussed together, and for all the motivation you shared we me. I remember several occasions where you dedicated your time – working hours, evenings and even weekends – to work with me on my thesis (and my thanks also goes to Nora for allowing you to do so!). Thank you so much for being a great researcher, mentor, motivator and friend!

My deepest gratitude goes to Christina Breitfuß, my incredibly smart and beautiful girlfriend. Thank you for motivating me again and again to push my work, and for understanding, tolerating and even encouraging me to spend countless weekends in front of my computer instead of with you. I'm so thankful that we were able to go to Sydney together and share this great experience with each other. Thanks for being my first reviewer and my last resort, and thanks for all the small (and big) things you do to make my life so much better. Your love and support mean everything to me!

Last but not least, I want to thank my family for supporting me through my whole life. You are my safe haven, my favorite holiday destination, and the reason why I came that far in the first place. Thank you so much for all you love and support!

Graz, 18.03.2018
Marco Steger

The research from DEWI (www.dewi-project.eu) leading to these results has received funding from the ARTEMIS Joint Undertaking under grant agreement No 621353. The authors acknowledge the financial support of the COMET K2 - Competence Centres for Excellent Technologies Programme of the Austrian Federal Ministry for Transport, Innovation and Technology (BMVIT), the Austrian Federal Ministry of Science, Research and Economy (BMWFW), the Austrian Research Promotion Agency (FFG), the Province of Styria and the Styrian Business Promotion Agency (SFG).

This work was also partially funded by the SCOTT (http://www.scott-project.eu) project. SCOTT has received funding from the Electronic Component Systems for European Leadership (ECSEL) Joint Undertaking under grant agreement No 737422. This joint undertaking receives support from the European Unions Horizon 2020 research and innovation programme and Austria, Spain, Finland, Ireland, Sweden, Germany, Poland, Portugal, Netherlands, Belgium, Norway. SCOTT is also funded by the Austrian Federal Ministry of Transport, Innovation and Technology (BMVIT) under the program "ICT of the Future" between May 2017 and April 2020. More information at https://iktderzukunft.at/en/. The authors also acknowledge the financial support of the COMET K2 Program of the Austrian Federal Ministries BMVIT and BMWFW, the Province of Styria, and the Styrian Business Promotion Agency (SFG).

Abstract

The complexity of automotive systems and especially of the used software has increased significantly in the last years due to the introduction of new safety mechanisms, entertainment features and connected services, as well as due to the installation of autonomous driving functions. This increased complexity of automotive systems and the consequent growth of the software embedded on several electronic control units within a modern vehicle raises the need for efficient mechanisms to update the embedded software. In this regard, the ability of efficiently performing software updates is beneficial over the entire lifecycle of a modern vehicle ranging from the vehicle development phase, over the assembly of the vehicle, to the maintenance of the vehicle once sold to the end user.

Traditional automotive software updates can be performed more efficiently by replacing the wired point-to-point links between dedicated diagnostic hardware providing the new software and the vehicle with a dedicated wireless network. Such a wireless network allows a user to update the software on several vehicles simultaneously.

Although beneficial and promising, wireless software update systems are particularly critical w.r.t. security, as they require access to all control units within a modern vehicle and thus are a worthwhile target for attacks. Recent attacks on automotive systems such as the cyber-attack on a Jeep Cherokee, where the hackers were able to remotely control the vehicle, have indeed shown the wide range of potential security threats and emphasize the need for comprehensive security concepts protecting automotive systems. Suitable security solutions must hence be in place to ensure the function of the vehicles, the integrity of the software, and especially the safety of the users.

In this thesis a solution for secure and efficient wireless software updates is proposed, with particular focus on local software update scenarios, where the wireless update is performed within a local environment such as the premises of a vehicle development company or within the building of a service center. The designed wireless software update system provides secure and efficient solutions to cover the entire software update procedure end-to-end, as it encompasses a secure software distribution mechanism based on Blockchain technology, allowing the vehicle manufacturer to securely distribute new software to local update providers such as a service center, as well as a framework for efficient and secure locally-performed software updates called EASE-UP. The latter provides advanced software update mechanisms such as parallel updates, where the software on several vehicles is updated at the same time, and partial updates, where only the differences between the currently installed and the new software needs to be transferred to the vehicle.

Using our testbed infrastructure with automotive EUCs we performed evaluations which show that both mechanisms are able to significantly reduce the overall update duration and thus to increase the efficiency of wireless software updates. Besides efficiency, we also tackle security-related issues within EASE-UP and provide a comprehensive security concept to tackle the peculiarities of different local software update environments. The security concept is designed using a measurable-security-based system design flow, where the required update system is analyzed, employed security methods are rated, and finally security requirements are extracted. These requirements are finally used to develop a security concept protecting EASE-UP by employing software- as well as hardware-based security features.

Zusammenfassung

Die Automobilindustrie hat sich in den letzten Jahren sehr stark gewandelt und der Entwicklungsfokus verschiebt sich, vor allem in Zeiten des automatisierten Fahrens, zusehends weg vom traditionellen Maschinenbau hin zum Einsatz von mehr und mehr Elektronik sowie verteilten Softwarearchitekturen. Die Komplexität der eingebetteten Software steigt dabei rasant an und auch die Fehlerhäufigkeit nimmt zu. Aus diesem Grund ist die Automobilindustrie heute mehr als je zuvor an innovativen Lösungen interessiert, die ein sicheres sowie schnelles Aktualisieren dieser Software während des gesamten Lebenszyklus eines Fahrzeugs erlauben.

Im Vergleich zu bereits existierenden Software-Update-Systemen, bei denen Diagnosehardware direkt über ein Kabel mit einem Fahrzeug verbunden wird, würden drahtlose Systeme, bei denen mehrere Fahrzeuge ohne den Einsatz von Kabeln mit neuer Software bespielt werden könnten, viel Zeit einsparen und damit einen signifikanten Vorteil bringen. Solche drahtlosen Systeme könnten bereits in der Fahrzeugfertigung eingesetzt werden um mehrere Fahrzeuge gleichzeitig mit der neuesten Software auszustatten, bevor diese die Fertigungsstraße verlassen. Auch in Werkstätten könnten solche Systeme vorteilhaft eingesetzt werden und es Mechanikern erlauben, zur gleichen Zeit an mehreren Fahrzeugen zu arbeiten.

Den angeführten Vorteilen von drahtlosen Lösungen stehen jedoch ernste Sicherheitsbedenken gegenüber. Software-Updates können alle Steuergeräte im Fahrzeug betreffen und daher muss Update-Systemen Vollzugriff auf alle Bereiche des Fahrzeuges gewährt werden. Bei mangelhaften Sicherheitskonzepten kann dies jedoch zu weitreichenden Bedrohungen führen. Erfolgreiche Cyber-Attacken auf vernetzte Fahrzeuge wie der Angriff auf einen Jeep Cherokee, wo es Hackern möglich war, das Fahrzeug aus der Ferne zu steuern, zeigen die enormen Risiken auf, die mit der Nutzung von drahtlosen Schnittstellen bzw. Systemen in modernen Fahrzeugen einhergehen. Geeignete Sicherheitskonzepte sind also unumgänglich um drahtlose Software-Update-Systeme in der Praxis einsetzen zu können.

In dieser Arbeit wird ein Gesamtsystem für sichere und effiziente drahtlose Software-Updates für Fahrzeuge vorgestellt. Der Fokus der Arbeit liegt dabei auf lokalen Update-Szenarien, in denen Software-Updates innerhalb eines Gebäudes bzw. eines abgeschlossenen Bereiches durchgeführt werden. Die entwickelte Lösung deckt dabei sowohl die Verteilung der Software, vom Fahrzeughersteller bis zu einem lokalen Software-Update-Anbieter wie zum Beispiel einer Werkstatt, sowie dem eigentlichen, lokal durchgeführten Update-Prozess ab. Die entwickelte Architektur zur Softwareverteilung nutzt dabei eine Blockchain-gestützte Netzwerkstruktur um die Integrität sowie die Echtheit der übertragenen Software zu garantieren.

Das Hauptaugenmerk der Arbeit liegt auf dem lokal durchgeführten Update-Prozess selbst. EASE-UP, ein drahtloses Software-Update-System für den Einsatz in verschiedenen lokalen Update-Szenarien, wurde entwickelt um diesen lokalen Update-Prozess möglichst sicher sowie effizient durchzuführen. EASE-UP setzt dabei auf fortgeschrittene Updateprozesse wie zum Beispiel parallele Updates, wo auf mehreren Fahrzeugen gleichzeitig neue Software installiert wird, oder partielle Updates, wo nur jene Teile der neuen Software, die sich von der aktuell am Steuergerät verwendeten Version unterscheiden, übertragen werden müssen. Zusätzlich wurde ein umfangreiches Sicherheitskonzept entwickelt, um EASE-UP und alle damit verbundenen Anwender, Geräte und Daten gleichermaßen zu schützen.

Die resultierende Gesamtlösung für lokal durchgeführte, drahtlose Software-Updates wird in der entwickelten Testumgebung entsprechend geprüft und auf ihre Eignung für den Einsatz in verschiedenen lokalen Update-Szenarien hin untersucht. Dabei werden auch die entwickelten Update-Mechanismen analysiert und miteinander verglichen.

Contents

List of Figures

List of Abbreviations

AES	Advanced Encryption Standard
AP	Access Point
BLE	Bluetooth Low Energy
CA	Certificate Authority
CAN	Controller Area Network
CGW	Central Gateway
CH	Cluster Head
CM	Cluster Members
COTS	Commercial Off-The-Shelf
CPU	Central Processing Unit
CRC	Cyclic Redundancy Check
DoS	Denial of Service
DT	Diagnostic Tester
DTC	Diagnostic Trouble Codes
DTM	Distributed Throughput Management
ECU	Electronic Control Units
GCM	Galois/Counter Mode
GUI	Graphical User Interface
HSM	Hardware Security Module
HWMP	Hybrid Wireless Mesh Protocol
IoT	Internet of Things
ITS	Intelligent Transportation System

JBC	Java Bouncy Castle
LAN	Local Area Network
LIN	Local Interconnect Network
LSB	Lightweight Scalable Blockchain
LSU	Local Software Update
MAC	Medium Access Control
MIC	Message Integrity Code
ODX	Open Diagnostic data eXchange
OEM	Original Equipment Manufacturer (i.e., car manufacturer)
OTA	Over-The-Air
PCB	Printed Circuit Board
PKI	Public Key Infrastructure
PoW	Proof of Work
RSA	Rivest-Shamir-Adleman; a public-key cryptosystem
RTT	Round Trip Time
SAE	Simultaneous Authentication of Equals
SHA	Secure Hash Algorithm
SPD	Security, Privacy, and Dependability
TC	Testbed Control PC
TN	Testbed Nodes
TPM	Trusted Platform Module
UDP	User Datagram Protocol
UDS	Unified Diagnostics Protocol
WSN	Wireless Sensor Network
WVI	Wireless Vehicle Interface
SE	Secure Element
V2V	Vehicle-to-Vehicle

Chapter 1

Introduction

The automotive industry has changed radically in the last decades and year, as basic mechanical engineering work is more and more replaced by electronics and software development. Due to this trend, the cost distribution for producing cars has significantly altered (i.e., in 2015, already 40% of the overall production costs of a car were dedicated to electronics and software) and new industrial players such as Google and Apple are starting to enter the automotive market [20].

The shift towards more and more electronics and software integrated in current as well as in future vehicles is driven by various factors including applications such as autonomous driving, an increased degree of connectivity, comfort features, and vehicle safety in general. All these functions heavily rely on electronic components (i.e., sensors and units) as well as on the embedded software running on these Electronic Control Units (ECU).

Already in 2009, first premium-class cars contained close to 100 million lines of software code executed on 70 to 100 ECUs [9]. The ECUs thereby fulfill different kind of tasks such as controlling the entertainment system of the car, the window lifters, the door locks, but also the engine as well as the transmission. As of today (2018), modern vehicles allow more and more autonomous driving functions and, because of that, ECUs have to handle even more complex and safety-critical tasks. System failures due to software bugs can indeed cause dangerous driving situations and lead to severe accidents.

One essential aspect, which is getting more and more important due to the rising amount of software code in vehicles, is the maintenance of the software running on automotive ECUs. This embedded software is very likely to change within the lifecycle of a vehicle, due to vehicle upgrades and added functionality, or due to the need of fixing bugs in the embedded software. Such software updates can already take place in the vehicle development phase, when engineers are testing new software versions for ECUs integrated in a test vehicle. Furthermore, software updates are required in the vehicle assembly line to install the latest software on one or several ECUs before a vehicle can be sold, as well as in vehicle service centers, when a vehicle upgrade is performed or a bug fix is required.

Today, dedicated and expensive equipment is utilized in all these scenarios to perform software updates. In most cases the equipment is connected to the vehicle by using its standardized diagnostic interface and via a wired point-to-point connection. Hence, current systems are expensive, heavy, inflexible, and do not allow to maintain the software of several vehicles at the same time. Replacing these wired point-to-point links with wireless ones can offer several advantages in all aforementioned scenarios.

1.1 Wireless Automotive Software Updates

Wireless software updates offer several advantages compared to wired update systems. In the following, the most important advantages are highlighted and briefly discussed.

Increased flexibility by utilizing wireless interconnections. When employing wireless networks instead of wired connections, the static point-to-point link between one vehicle and the update device – the so-called Diagnostic Tester (DT) – can be replaced by a wireless network connecting all vehicles in close proximity (e.g., within a building such as a service center) to the DT and other handheld devices such as tablets utilized by the involved users (e.g., mechanics in a typical service center scenario). The resulting *interconnectivity between all involved entities* (i.e., vehicles, DT, and users utilizing tablets) significantly increases the flexibility of the update system: *a user can connect to several vehicles* in a row without unplugging from one vehicle to connect to a second one, as it would be the case with wired systems.

Increased update process efficiency. The additional flexibility gained by employing wireless systems will increase the efficiency of the process (i.e., the involved users will save time). Furthermore, such wireless systems will also allow to perform *wireless software updates for several vehicles in parallel*, *significantly reducing the time required to maintain vehicles* (e.g., in a service center). Such parallel software updates can also be very beneficial in assembly line scenarios, where potentially a lot of vehicles will require the identical software update. In such highly automated scenarios (where every second that can be spared leads to higher profits), reduced software update durations would decrease the production costs. For all mentioned software updates scenarios, efficiency (i.e., reducing the overall update duration) is essential and can be significantly increased by employing wireless systems.

Autonomy of the involved entities. Future vehicles will be more and more connected and thus equipped with different wireless interfaces. These interfaces could potentially be used to perform wireless software updates in a very autonomous way: while current vehicles still need a wireless dongle plugged-in via a diagnostic vehicle interface to connect to a wireless update system, future vehicles could just use the built-in interfaces to connect to such a system. This increased *autonomy* would again be very advantageous for all aforementioned software update scenarios and thereby especially relevant when updating a vehicle on the assembly line.

Possibility to cover a vehicle's entire lifecycle. Wireless software update systems can significantly increase the *flexibility*, *autonomy*, and thus the overall *efficiency* of automotive software updates. The latter can be utilized in different update scenarios and therefore support the entire lifetime of a modern vehicle: from vehicle development and vehicle assembly (i.e., before a vehicle is sold) to vehicle maintenance (i.e., after a vehicle is purchased), where necessary updates (e.g., due to a required bug fix or to upgrade the ECU's functionality) are performed locally in a service center or carried out using a remote connection to the vehicle (especially relevant for future vehicles).

Remote updates will allow to install new software on vehicles already out in the field by utilizing suitable Internet links such as a 3G connection between the vehicle and the OEM, as shown in Figure 1.1. Such wireless remote or Over-The-Air (OTA) updates are currently investigated and addressed by various researchers as well as the automotive industry [36, 49, 50], and allow to remotely install new software on a vehicle through a connection to the Internet. Tesla was the first car manufacturer (OEM) enabling and allowing automotive remote updates for an entire fleet of vehicles since 2014 [19].

In future, remote updates will be used by all OEMs as they allow to perform certain software updates without the need of large vehicle recalls. However, a large portion of automotive software updates will still be performed in local environments (i.e., the update is done within a dedicated area or building), where vehicles are connected to diagnostic equipment (i.e., the DT and handhelds running dedicated apps) using a wireless network such as Wi-Fi. The entire update process is carried out locally without a direct involvement of the OEM by utilizing secure and efficient wireless software update systems.

1.2 Locally-performed Wireless Software Updates

The use of secure and efficient wireless software update systems will be beneficial in different Local Software Update (LSU) scenarios: in the vehicle development phase, development engineers can seamlessly install test software on several test vehicles/ECUs at the same time without physically (i.e., by using a cable) connecting to each of them. Parallel software updates can also be performed in the vehicle assembly line as well as in service centers. Mechanics in such a service center will additionally benefit from parallel updates, as they can work on several vehicles at the same time using a single handheld device. For example, a mechanic can first trigger a software update on two vehicles (i.e., a parallel update, where the same software is installed on multiple ECUs simultaneously) and, while this update is carried out, diagnose and repair other vehicles.

LSUs versus OTA updates. One of the main differences between OTA updates and LSUs is the *interconnection between the OEM and the vehicle* during an update, as sketched in Figure 1.1. While this connection is essential for remote updates to transfer new software to the vehicle, but also to handle the authorization of the update on vehicle/ECU level, LSUs typically do not require any direct interaction with the OEM at all during the update process, as the required authorization keys and the new software are stored on a dedicated server within the local software provider (e.g., a service center). A dedicated *software distribution step* is used in LSU scenarios to transfer new software plus the required authorization keys to different local software update providers.

This *software distribution step* represents another difference between OTA updates and locally-performed updates. An end-to-end update process in LSU scenarios encompasses two dedicated (and discontiguous) steps as shown in Figure 1.1: step 1, the software distribution and step 2, the local software installation, where new software is transferred from the DT to the vehicle and then installed on the ECU. Typically, step 1 and step 2 are not directly connected, meaning that the software distribution is done once new software is available[1]. An actual update (i.e., step 2) is performed once a vehicle demanding the new software is maintained within a local software provider (e.g., days after the new software distributed by the OEM was received). OTA updates, in contrast, are only performed when there is an actual need for the update: a vehicle periodically checks for new software.

[1] In the early days, this distribution step was done by sending out DVDs, today a secured Internet connection is utilized.

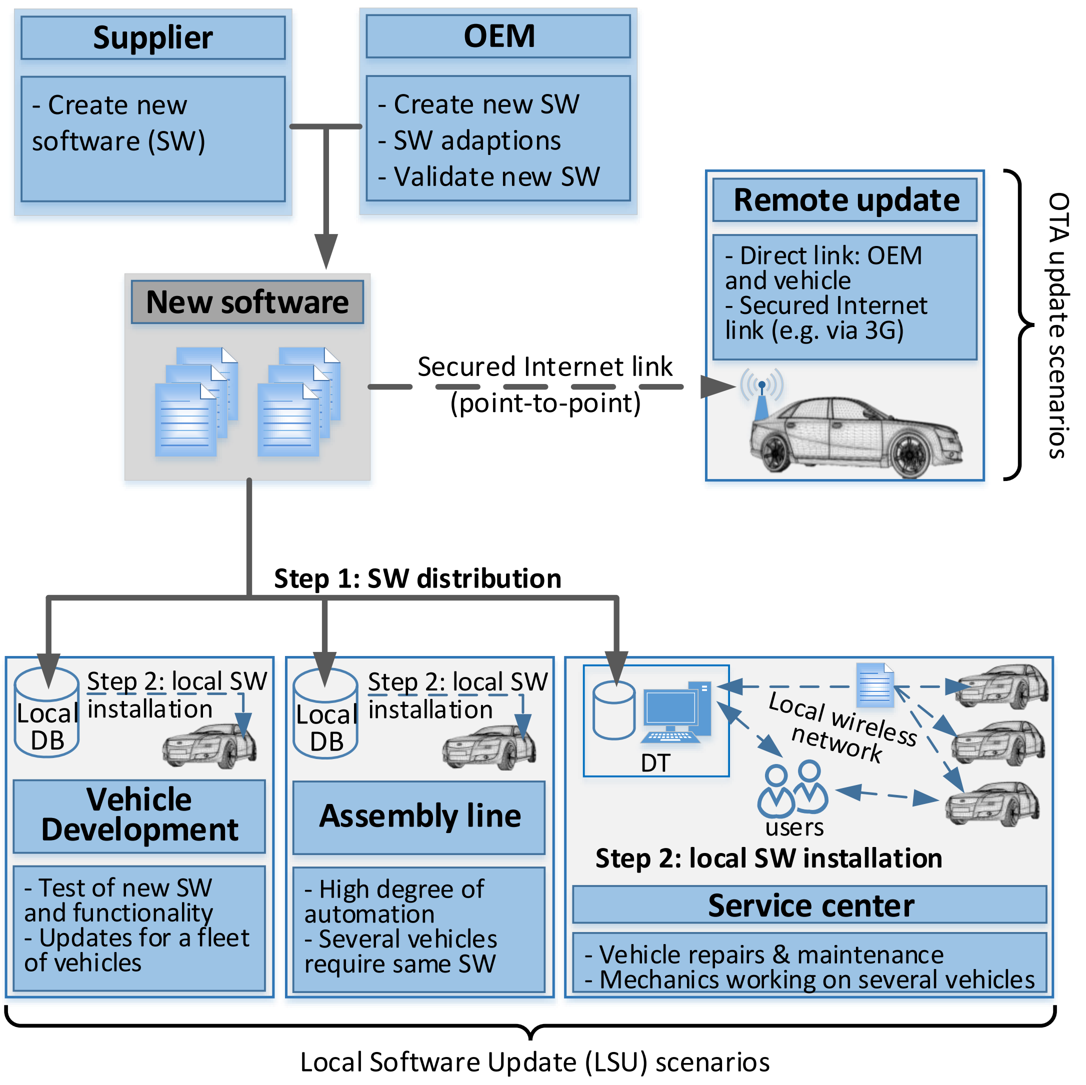

Supplier
- Create new software (SW)
OEM
- Create new SW
- SW adaptions
- Validate new SW
New software
Secured Internet link (point-to-point)
Remote update
- Direct link: OEM and vehicle
- Secured Internet link (e.g. via 3G)
OTA update scenarios
Step 1: SW distribution
Local DB
Step 2: local SW installation
Vehicle Development
- Test of new SW and functionality
- Updates for a fleet of vehicles
Local DB
Step 2: local SW installation
Assembly line
- High degree of automation
- Several vehicles require same SW
DT
Local wireless network
users
Step 2: local SW installation
Service center
- Vehicle repairs & maintenance
- Mechanics working on several vehicles
Local Software Update (LSU) scenarios

Figure 1.1: Different wireless software updates scenarios. New software can be installed on the ECUs remotely by employing an Internet link as well as using a local wireless network.

Once new software is available, a direct secured link between the vehicle and the OEM is established, the software is transferred to the vehicle, and finally installed on the ECU.

The third difference between an OTA and an LSU system is *the scope of such a wireless software update system*. While *OTA updates are only targeting modern vehicles* equipped with the required wireless interface (e.g., a dedicated 3G/4G module), *LSUs are used to update the software on all kind of vehicles* including prototype/test vehicles (i.e., vehicle development), new vehicles (i.e., within the vehicle assembly line), as well as used vehicles (i.e., in a service center) of different ages. Especially in service centers but also in vehicle development performed by supplier companies, an LSU system will provide software updates for vehicles from different OEMs. OTA updates, in contrast, will typically only target recently produced vehicles (e.g., not older than two years) of the same brand/OEM.

1.3 Problem Statement

The primary aim of this doctoral thesis is to develop an *end-to-end solution for locally performed wireless software updates*, while remote/OTA updates are not in scope of this work. The focus on locally-performed software updates is mainly due to high potential of improving the efficiency of vehicle maintenance in LSU scenarios: a well-designed solution will help to significantly decrease the time required to perform a software update for one or several vehicles and hence raise the profit for the local software update provider as well as increase the satisfaction of the involved users (i.e., both mechanics/engineers performing the update, as well as the vehicle owner).

The developed system shall be able to fulfill the requirements and the peculiarities of all LSU scenarios, and especially focus on the essential aspects that are discussed next: *security* and *efficiency*. Furthermore, also the *software distribution step*, where new software is sent from the OEM to the local software update providers, shall be addressed, leading to a secure end-to-end solution for LSUs.

Securing wireless automotive LSUs. Security is a key aspect of a wireless automotive software update system, as the use of wireless networks as basis for LSUs as well as the integration of wireless interfaces in (future) vehicles introduces a range of new security threats. A malicious attacker could for example try to interfere a running software update to slow down the entire update process, to install malicious software on the ECU of a vehicle by tampering with the transferred data, or steal expensive software by eavesdropping the wireless channel. First real-world attacks where hackers exploit insufficiently secured wireless interfaces to gain remote control over a Tesla [15], a Corvette [22], or a Jeep [63], show the huge potential of hacking connected vehicles.

A wide range of potential security threats are mainly affecting the user (i.e., his/her safety and privacy), the confidentiality and genuineness of the software, required authentication and authorization keys (e.g., the authorization key required to perform a software update on a specific ECU), as well as the integrity of the vehicle and all other devices (e.g., DT). One of the goals of this doctoral thesis is to *analyze these threats* and to define a suitable *security concept that can efficiently mitigate them.*

A key challenge in having a general security solution is that LSU are performed in different environments by different users and have thus various requirements w.r.t. security. A generic security concept covering all these scenarios must therefore be designed in a way that it can be adapted to the needs of the actual scenario and the corresponding security level that is required. Furthermore, a structured design process shall be identified and followed when developing such a system. The chosen design flow shall be similar to the well-established automotive functional safety processes defined in the ISO 26262 standard [33], as these processes are well known and accepted within the automotive industry. Hence, within this doctoral thesis, a *structured security design approach* will be defined in a way that different LSU scenarios, as well as the corresponding environments and *levels of security*, can be taken into account when creating the *final security concept.*

Making wireless automotive LSUs more efficient. Besides security, also efficiency is an essential aspect of wireless automotive software updates. With regard to such updates, efficiency typically refers to the ability of handling/performing the entire software update procedure of one or several vehicles as fast as possible. In LSU scenarios, efficiency can be achieved on multiple layers:

The network layer (i.e., the wireless communication network chosen to interconnect vehicles, users, and diagnostic equipment) have a significant impact on the reliability and in further consequence the efficiency of the entire system. Weak links, high latency, or a lack of scalability will decrease the update efficiency (i.e., due to a slow wireless data transfer, the update duration will be prolonged), while redundancy and high bandwidth will help to decrease the overall update duration. Additionally, the harsh environment in which automotive software updates are performed, must be taken into account. Therefore, another goal of this thesis is to *analyze available wireless protocols* and choose the *most suitable candidate* to be used for a wireless automotive LSU system.

On the application layer, different factors such as the employed security mechanisms, the chosen diagnostic protocol, as well as the utilized software update mechanism have a significant impact on the efficiency of an automotive LSU system. Thus, this doctoral thesis will *investigate and develop different software update mechanisms* increasing the efficiency of the entire update process (i.e., decrease the overall update duration), and the related security concept will be designed to support these mechanisms.

Ability to comprehensively test systems for wireless automotive LSUs. The aforementioned factors (i.e., the employed security mechanisms, the used diagnostic protocol, and the utilized software update mechanism) indeed influence the duration of a software update and hence *their impact on the efficiency* of a system must be evaluated. This is especially important w.r.t. the employed security concept, as security and efficiency often strongly influence each other. Hence, security, efficiency and other related aspects do not only need to be considered together when designing and implementing an end-to-end solution for LSUs, but also when it comes to evaluating and testing the developed system.

To perform such evaluations, an extensive testbed infrastructure is required. However, as of today, neither a suitable testbed is available nor its possible design is described in related works. Therefore, this doctoral thesis will focus on the development of a *testbed infrastructure* allowing to evaluate various important aspects (e.g., impact of utilized security measures and advanced update mechanisms on the software update duration) as well as different system configurations within one testbed.

End-to-end solution for wireless automotive LSUs. The software update procedure for LSUs encompasses two individual steps, as illustrated in Figure 1.1: the software update distribution step (step 1; where new software is sent by the OEM and/or its suppliers to local software update providers such as a service center), and the actual software installation process done within a local environment (i.e., step 2). Both steps must be secured with suitable security measures and shall be based on an efficient architecture allowing a fast software distribution as well as a quick software installation.

The ultimate goal of this thesis is hence the *development of a secure and efficient end-to-end solution for automotive wireless LSUs* covering both aforementioned steps: the design, implementation, and evaluation of a framework allowing efficient as well as secure locally-performed software updates, and the definition of a suitable software distribution architecture providing an efficient as well as secure channel between the OEM and the LSU providers.

1.4 Contributions

In this section the scientific contributions of this doctoral thesis in the area of secure and efficient wireless automotive LSUs are summarized and references to the corresponding publications are given.

End-to-end solution for LSUs. In this thesis an end-to-end solution for locally-performed wireless automotive software updates is proposed. This solution encompasses a secure software distribution architecture [59] as well as a framework for efficient and secure software updates within different LSU scenarios [37, 61]. Special focus is put on the local software update procedure handled within a dedicated area or building such as a vehicle assembly line or a service center.

Secure and efficient software distribution based on Blockchain technology. Before new software can be installed on an ECU of a vehicle within a LSU scenario, the software first needs to be distributed from the OEM to all concerned local software update providers such as a service center. This distribution chain includes the transfer of the new software from a supplier (i.e., actually creating the new binary) to the OEM, the verification of the binary by an OEM and, finally, the distribution of this binary to local software providers (i.e., the considered LSU scenarios) as well as to smart vehicles remotely. As an important contribution of this thesis w.r.t. the targeted end-to-end solution, we propose a Blockchain-based security architecture allowing trustworthy software distribution [59]. The developed architecture covers both steps of the distribution chain (i.e., step 1: from supplier to OEM; step2: from OEM to the LSU environment), and can be connected to the developed framework for secure and efficient wireless automotive software updates.

Comprehensive framework for secure and efficient wireless LSUs. The main contribution of this thesis is a framework for locally-performed **E**fficient **A**nd **SE**cure wireless automotive software **UP**dates (EASE-UP). EASE-UP combines the defined security concept [57] with advanced software updates mechanisms [37] and the secure as well as reliable wireless network infrastructure [60] to build a comprehensive LSU framework. The latter allows to carry out efficient and secure wireless software updates in all aforementioned LSU scenarios. The developed framework is fully implemented [37] and the resulting prototypes can be used to demonstrate both wireless vehicle diagnostics as well as wireless LSUs.

Selection of an efficient and secure wireless protocol for LSUs. EASE-UP's update framework requires a wireless medium used to interconnect all entities involved in a typical LSU scenario. This wireless medium has a significant impact on the security as well as the efficiency (i.e., reliable and fast networks leading to a decreased update duration) of EASE-UP. Therefore, one contribution of this thesis encompasses the identification of requirements w.r.t. a wireless medium for automotive LSUs, and the comparison of different wireless protocols w.r.t. their ability to fulfill all these requirements. The performed investigation shows the applicability of IEEE 802.11s as wireless medium for wireless automotive software updates [60], as well as leads to the choice of IEEE 802.11s [29] as the most promising candidate for a LSU framework.

Securing EASE-UP. Security is a key aspect of EASE-UP, as a suitable wireless LSU system potentially provides numerous attack vectors that can be exploited by hackers (i.e., external attackers) as well as malicious users (i.e., internal attackers). To mitigate the risk of both internal and external attacks, suitable security measures must be employed, while also considering the requirements of different LSU scenarios. The latter will demand different security levels, as the updates are performed in diverse environments by different types of users.

Methodology based on measurable security for automotive system design. A security concept shall be configurable in a way that it provides sufficient security measures while not disproportionally slowing down the entire system (i.e., decreasing the efficiency). To design such a concept, a structured design approach is required. Thus, an important contribution of this thesis is the selection and refinement of a security design approach for automotive systems based on a measurable security framework [62]. The developed design approach allows to analyze a system w.r.t. different application scenarios (e.g., different local software update scenarios) and results in security requirements as well as possible system configurations for each considered scenario. These results are the basis for the definition of a suitable security concept.

Generic security concept for wireless automotive software updates. For each local software update scenario specific peculiarities such as the education of the involved users or the environment must be considered and hence a certain security level is required. Indeed, one can therefore develop a tailor-made system for every LSU use case. However, we rather aim to design a generic security concept for EASE-UP which can be utilized in all LSU scenarios. Thus, a further contribution of this thesis is the definition of a generic security concept for automotive software updates [57]. The definition procedure complies to the design workflow of SAE-J3061 [52], a new security standard for vehicles [62].

Evaluation of EASE-UP. The developed update framework combines various aspects of security and efficiency, relies on different protocols (e.g., diagnostic, wireless, and vehicle-specific protocols and standards), as well as encompasses several interconnected entities (e.g., DT, handheld device, wireless vehicle interface), while taking the specific requirements and aspects of the different LSU scenarios into account. Therefore, EASE-UP's implementation is rather complex and must be extensively tested to ensure that software updates are performed in a secure, efficient and reliable way.

Comprehensive testbed infrastructure. Due to the lack of existing suitable testbed environments, a comprehensive testbed architecture allowing to analyze the impact of different application scenarios, network topologies and environments on the efficiency of wireless automotive software updates was developed [58]. The resulting testbed infrastructure allows to emulate different LSU scenarios and to perform detailed performance evaluations.

Performance evaluation. The last contribution of this thesis is a detailed performance analysis of EASE-UP's features. The gathered results encompass i) an analysis of the impact of different security measures on the local software update duration [37, 58], ii) an evaluation of the developed wireless software update mechanisms [58], and iii) an analysis of the performance of the developed end-to-end solution [59], where the duration of the software distribution is compared with the latency added by the local update process.

1.5 Structure

The remainder of this dissertation is organized as follows. Chapter 2 provides background information describing automotive standards and diagnostic protocols. Furthermore, an overview on Blockchain technology is given, an automotive security standard providing basic information about how to design a secure automotive system (i.e., SAE J3061 [52]) is described, and a measurable security design approach, namely the SHIELD multi-metrics [21], is discussed. Thereafter, in Chapter 3, existing work in the area of automotive software updates is presented and the related challenges with specific focus on security and efficiency are highlighted.

In Chapter 4, the developed end-to-end solution is presented. This end-to-end solution encompasses the Blockchain-based security architecture for secure software distribution, as well as EASE-UP's architecture allowing secure and efficient wireless automotive LSUs. Thereafter, in Chapter 5, networking aspects w.r.t. EASE-UP are discussed and the defined framework is explained in more detail. In particular, different wireless protocols are first analyzed w.r.t. their applicability for wireless software updates and the most suitable candidate is selected. Furthermore, the defined wireless update protocol is described and the developed advanced software update features are explained. Chapter 6 is dedicated to the security-related aspects of EASE-UP. In this chapter, the developed structured system design approach as well as the defined security concept for locally-performed wireless automotive software updates are described. In Chapter 7 the developed testbed infrastructure is described and, thereafter, the performed measurements and experiments are discussed. Thereby, a detailed evaluation of the efficiency of EASE-UP and its key features (i.e., the developed update mechanisms and the defined security concept) is presented. The gathered results show the benefits of employing advanced software update mechanisms and prove the efficiency of the developed LSU system compared to a wired solution. Chapter 8 concludes this doctoral thesis by summarizing the obtained results beyond the state of the art, by discussing limitations, and by providing an outlook on future work.

Chapter 2

Background

In this chapter background information is presented on which thesis builds. In particular, information about different in-vehicle communication protocols as well as automotive diagnostic standards is provided in Section 2.2, an overview of the Blockchain technology is given in Section 2.3, in Section 2.4 the new SAE security standard is described, and a measurable security system design approach is presented in Section 2.5. Please feel free to skip any or all of these sections in case you have sufficient knowledge in the discussed fields. Links to the given background information are provided throughout the thesis.

2.1 Automotive ECUs and In-vehicle Communication

A modern vehicle employs dozens of ECUs interconnected by its in-vehicle communication systems to perform various tasks ranging from controlling the engine, the brakes, or the steering system, to running the built-in navigation and infotainment systems. In a typical automotive in-vehicle communication system, ECUs are clustered according to their dedicated task, as shown in Figure 2.1 and described in [24, 31].

Each cluster is interconnected using a dedicated automotive bus such as Controller Area Network (CAN), FlexRay, or automotive Ethernet [24]. This bus allows ECUs within a cluster to directly communicate with each other. Furthermore, each bus is also connected to a Central Gateway (CGW) interconnecting the single bus systems and thereby forming the in-vehicle communication system of a modern vehicle. The CGW is an essential device as it enables distributed applications where ECUs of different clusters are collaborating while protecting the vehicle and its ECUs from unauthorized access and external attacks. Thus, this crucial device is broadly investigated by academy and industry (e.g., see [31]).

In most vehicles, the CGW separates the internal bus systems from the diagnostic interface used by external entities (e.g., a DT within a service center) to access the in-vehicle communication system. The access management of the CGW makes sure that only authorized entities are allowed to interact with the integrated ECUs of a vehicle or to read/write from/to the internal bus systems. Authorized entities can use the diagnostic interface and automotive diagnostic protocols to query the vehicle for Diagnostic Trouble Codes (DTC) and to perform software updates for the ECUs. In the following some of these protocols are described in more detail.

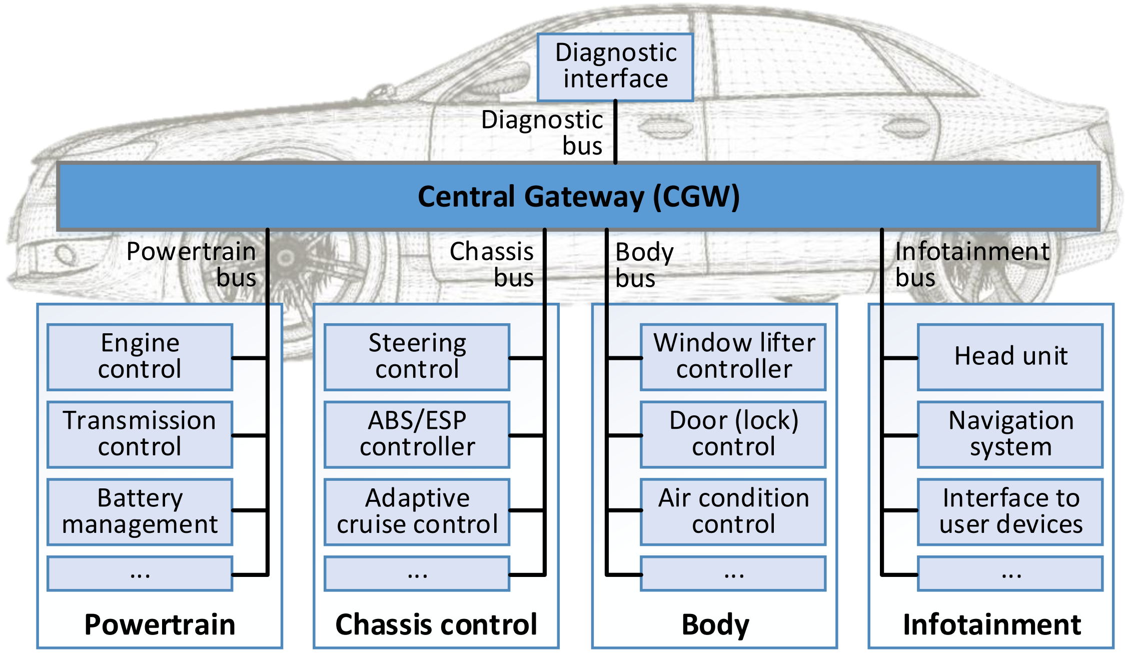


Figure 2.1: ECUs are clustered according to their task within the vehicle. Different bus systems are used to interconnect the ECUs within a cluster and a Central Gateway (CGW) is employed to allow the communication between ECUs of different clusters [31].

2.2 Automotive Standards and Protocols

Several automotive standards are supporting the software update process or are describing the underlying communication protocols. In the following, the most important standards are addressed and an overview on the automotive diagnostic stack is given.

2.2.1 Open Diagnostic data eXchange (ODX)

ODX [32] is an automotive data exchange format based on XML providing standardized mechanisms to exchange diagnostic-related information between vehicle, ECU, and tool (i.e., DT) manufacturer. An ODX file contains information about the vehicle and its configuration (e.g., any optional features), as well as information about all built-in ECUs and how they are interconnected within the vehicle (i.e., bus type and ECU IDs). Software updates are seen as an essential part of vehicle diagnostics and hence ODX files also include information about the software available for each ECU.

In Figure 2.2 an overview on the ODX file format is given. A typical ODX file will contain information about i) the vehicle itself including physical as well as logical links between ECU (i.e., VEHICLE-INFO-SPEC), ii) the available diagnostic protocols including defined request and the corresponding response messages (i.e., DIAG-LAYER-CONTAINER), iii) the communication specification including the employed communication protocols (e.g., CAN), and the corresponding communication IDs (e.g., the CAN IDs of all ECUs connected by the bus) and parameters (i.e., COMPARAM-SPEC), iv) available multi-ECU jobs (i.e., MULTI-ECU-JOB-SPEC; typical not used in practice), and v) how to program ECUs including information about the ECU's memory layout, a software version identifier, and information about each single block of a program.

The developed wireless automotive software update system relies on different information contained within a typical ODX file and hence supports this file format (i.e., a parser to retrieve all required information was developed). The DT will store ODX files in a local database or request an ODX file from the OEM via a dedicated backbone network if no ODX description is available in the local database. Please note that ODX files will be parsed by the DT and that only extracted (and required) data is forwarded to the WVI/handheld, as ODX files can be quite large in size and contain lots of non-relevant information (for the update process).

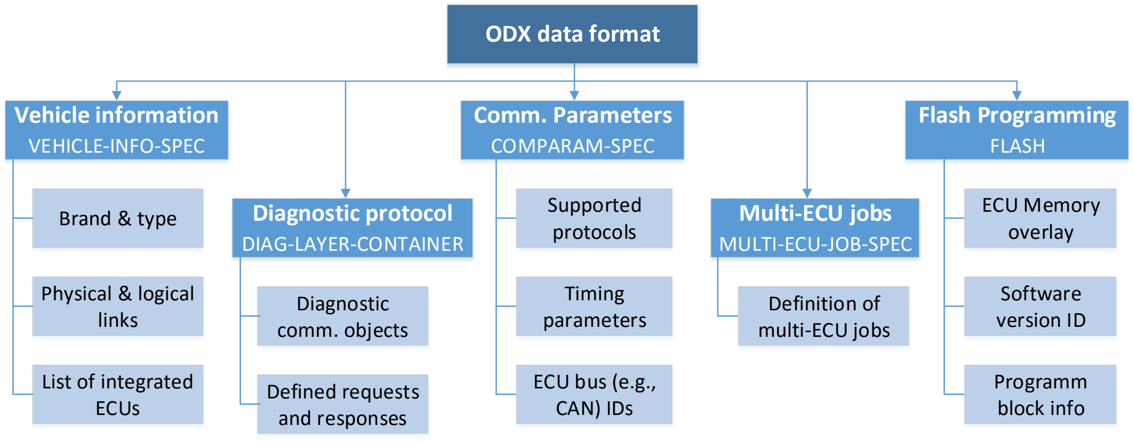


Figure 2.2: ODX data model. An ODX file contains information about the vehicle, the available diagnostic and underlying communication protocols, as well as the flash process required to update the software of an ECU.

2.2.2 Protocol Stack for In-Vehicle Usage

Wired vehicle diagnostics (i.e., direct, point-to-point connection between the DT and the vehicle) including software updates for ECUs are already well specified in the automotive domain and several standards describe how this communication is carried out on each layer. In Figure 2.3 an overview of these layers is given. The wireless software update system developed within this thesis also supports these standardized protocols as they allow to communicate with a vehicle and its ECUs in an OEM-independent as well as backward-compatible way. In particular, the developed WVI supports OBD and CAN connections (i.e., the transport layer as well as the underlying physical interfaces) to the vehicle and employs Unified Diagnostics Protocol (UDS) to handle the software update process between the WVI and the concerned ECU. More insights on the UDS protocol are given in the next section.

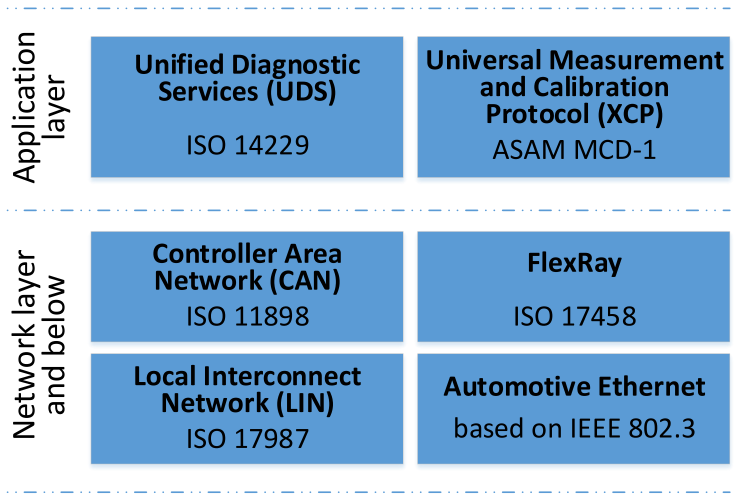

Application layer
Unified Diagnostic Services (UDS)
ISO 14229
Universal Measurement and Calibration Protocol (XCP)
ASAM MCD-1
Network layer and below
Controller Area Network (CAN)
ISO 11898
FlexRay
ISO 17458
Local Interconnect Network (LIN)
ISO 17987
Automotive Ethernet
based on IEEE 802.3

Figure 2.3: Automotive communication and diagnostic protocol stack. Several candidates for automotive communication protocols (often covering the physical, data link, and network layer) as well as diagnostic protocols are available.

2.2.3 Unified Diagnostics Protocol (UDS)

The UDS protocol is defined in ISO 14229 [1] and covers almost all aspects (including ECU programming) of vehicle diagnostics. It was initially specified for the use on top of UDS, but there are also extensions for other underlying protocols such as FlexRay or automotive Ethernet. UDS is supported by a wide range of current ECUs and an UDS stack is often integrated in the bootloader of the ECU to provide a standardized way to update the software running on it. Hence, the developed software update system (and in particular the WVI) uses UDS as primary update protocol within the vehicle (i.e., to transfer data from the WVI to the concerned ECU via CAN). However, also other diagnostic as well as communication protocols can be integrated easily due to well-defined interfaces between the core software components of the WVI.

A UDS-enabled ECU supports different sessions: in normal/operation mode (i.e., UDS *Default Session*), the ECU supports a basic set of diagnostic requests. For more comprehensive diagnostics (i.e., UDS *Extended Diagnostic Session*) and to perform an ECU programming process (i.e., UDS *Programming Session*), UDS also allows special sessions and provides well-defined mechanisms to switch between the default session and any other diagnostic session. The UDS session management is implemented as state machine as sketched in Figure 2.4. Typically, an authorization step based on a Seed & Key mechanism is required to approve any special session (i.e., authorize the session-specific requests). Please note that a Seed & Key mechanism is rather weak compared to other available security measures. Hence, future ECUs will potentially be equipped with secure elements (i.e., dedicated hardware security chips) allowing to employ more sophisticated security mechanisms (see [41] or [49]).

UDS is following a typical request-response communication scheme, where most of the defined requests and the corresponding responses are dedicated to a specific session. In the programming session the following requests are specified in UDS (please note that

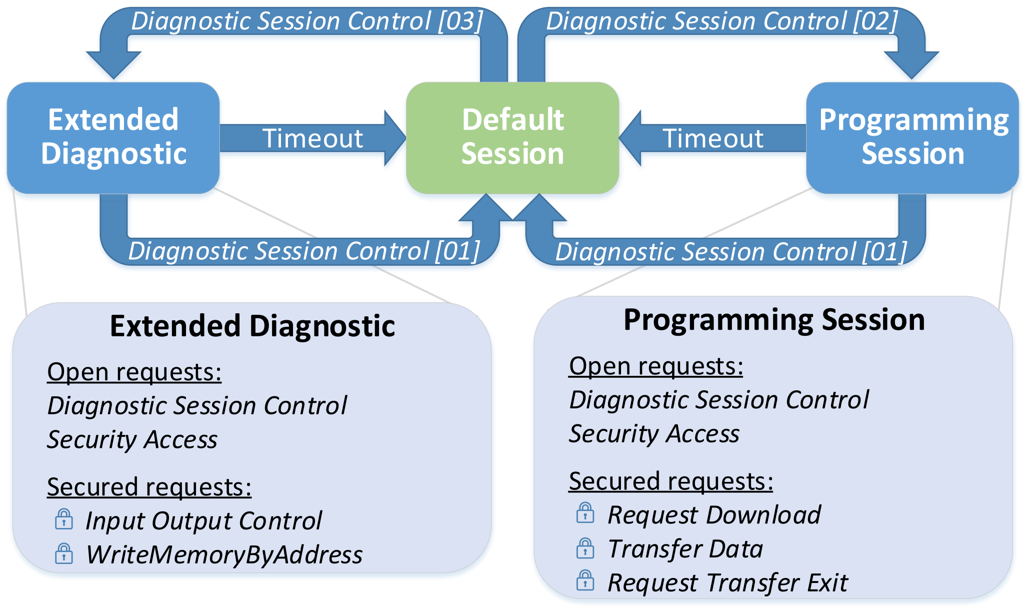


Figure 2.4: UDS session management. The *Default Session* is automatically launched after an ECU reset and once an advanced session times out. UDS *Security Access* is used to authorize critical operations such as ECU programming.

UDS allows using additional OEM-specific requests):

- ***Request Download***: initialize a data download to the ECU. This request is typically used to inform the ECU about the current data block to be transferred (e.g., its start address and size). This command will be used multiple times within a programming session, if more than one code segment shall be transferred.
- ***Request Upload***: initialize a data transfer from an ECU to the DT. This function can be used to read data segments from the ECU memory.
- ***Transfer Data***: the data transfer itself. The request includes a sequence number (one byte) followed by the bytes to be transferred.
- ***Request Transfer Exit***: to finalize the data transfer (i.e., upload or download).

It is important to note that there is no dedicated software installation process involved on ECU level, as the new software is directly written into the flash memory of the ECU (i.e., using the *Transfer Data* request) and executed after an ECU reset (i.e., UDS request *ECU Reset*). The OEM or the ECU manufacturer can also develop dedicated validation routines on the ECUs to verify an update (i.e., using the UDS *Routine Control* request).

2.3 Blockchain Technology

Blockchain is a distributed database (often referred to as ledger) maintaining a continuously growing list of blocks that are chained to each other [13]. In 2008 the Blockchain technology was proposed as the underlying platform of the first cryptocurrency Bitcoin by Satoshi Nakamoto [42]. Today (2018), Blockchains are not only employed in various cryptocurrencies, but also in other applications such as tracking of goods and valuables (e.g., for diamonds [34]), as well as in e-governance and the finance sector.

2.3.1 How Blockchain Works

The basic structure of a Blockchain is shown in Figure 2.5. A block within the Blockchain contains a predefined number of transactions, the smallest piece within a Blockchain, and includes metadata about a specific event. In Bitcoin, a transaction describes a transfer of X bitcoins from user A to user B. Each transaction as well as every block is labeled with a unique identifier (ID). This ID is created by computing the hash value of the content of a transaction or block, respectively. The actual chaining of blocks is done using these IDs. A new block contains, besides the transactions and other metadata, also the ID of the latest added block of the Blockchain. Before the new block is added to the Blockchain, its ID is computed by hashing the content of the transactions, the meta data, and the previous block ID. Hence, a tamper-proof chain of blocks is created as every change within a chained transaction or a stored ID will be immediately detected (as the hash value will change when only one bit is changed within a block/transaction).

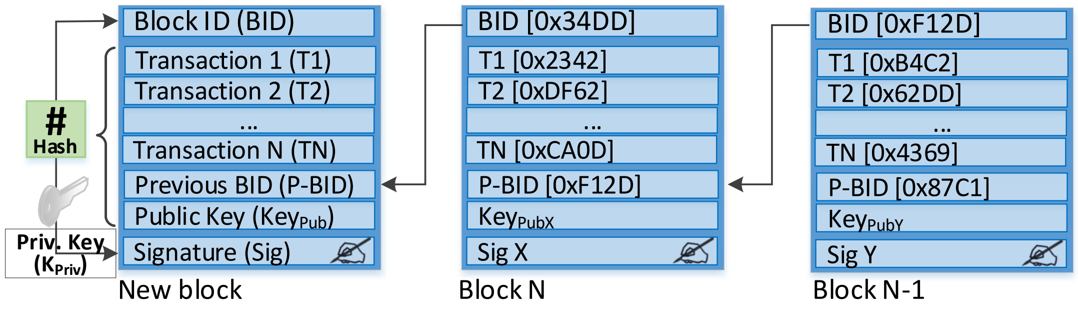


Figure 2.5: Example of a local Blockchain instance. Blocks containing transaction are chained together using the previous block ID.

Blockchain is managed distributedly by a peer-to-peer network, where each node and user is identified using a public key. Blockchain networks can be public allowing users from around the world to join the Blockchain ecosystem, or they may also be private (e.g., when used internally within a corporation with affiliates around the globe) and thus only be accessible by authorized users and entities. In both approaches (i.e., public and private Blockchains), a new transaction is signed by its creator using a digital signature (e.g., RSA) and then broadcast to the entire network. In the next step, all nodes within the network will verify this transaction by validating the signature of the transaction generator. The same principle applies for new blocks except that the other nodes will not only verify the signature of the block creator, but also all contained transactions.

Classical Blockchain architectures suffer from high (processing and packet) overhead as well as low scalability and throughput. Especially the *proof-of-work* (PoW) employed by such architectures is critical when the Blockchain shall be used within resource-constrained environments such as the Internet of Things (IoT), but also when using it within automotive applications. The PoW is a part of the Blockchain consensus algorithm that is used to add a new block in the Blockchain. When a new block shall be added, all Blockchain nodes (the so-called miners) will compete with each other solving a complex cryptographic puzzle. This requires a lot of resources (i.e., CPU, power, as well as time) and hence is not suitable for resource-constrained entities. Dorri et al. proposed a light-weight Blockchain approach called Lightweight Scalable Blockchain (LSB) dedicated to IoT applications [12], where the PoW is replaced by a scheduled block generation process. As a consequence, LSB eliminates the significant processing overhead of conventional Blockchains and is optimized for the IoT and other large-scale low-resource networks.

2.3.2 Lightweight Scalable Blockchain

LSB is based on a hierarchical communication model where the network is divided into clusters that distributedly manage the public Blockchain. Each cluster consists of numerous Cluster Members (CM) and is managed by one Cluster Head (CH). CHs are maintaining a local copy of the Blockchain and are interconnected with other CHs by the overlay network. The mostly resource-constrained CMs do not have to maintain such a Blockchain copy, but can only access other clusters via their CH and the overlay network. Thus, a new transaction created by a CM will first be sent to the CH and then broadcast to the other CHs using the overlay.

The LSB architecture is able to fulfill the aforementioned requirements of a trustable software distribution. Hence, a slightly adapted version of LSB is used as the basis for the designed automotive security architecture allowing secure and efficient software distribution. This architecture is described in Section 4.4.

2.4 The SAE J3061 Automotive Security Standard

In January 2016, SAE released the first automotive security standard called *Cybersecurity Guidebook for Cyber-Physical Vehicle Systems* [52]. This standard describes best practices intended to be flexible, pragmatic, and adaptable in their further application to the vehicle industry as well as to other cyber-physical vehicle systems. The defined methods and procedures as well as the characteristic V-model (i.e., development lifecycle) are based on the well-established automotive functional safety standard ISO 26262 [33].

The standard defines a complete lifecycle process framework that can be tailored and utilized within each organization's development processes to incorporate cyber-security into cyber-physical vehicle systems from concept phase through production, operation, service, to decommissioning [62]. Chapter 6 (*Cyber-security process overview*) is of most interest for the definition of the DEWI[1] Security Metric (i.e., the defined measurable security design approach; see Section 6.1.1) as it describes the flow of activities in the concept phase as shown in Figure 2.6. Especially the steps i) *Threat Analysis and Risk Assessment (TARA)*, ii) *Cybersecurity Concept*, and iii) *Identify Cybersecurity Requirements* are particularly important. The TARA is employed to identify and assess potential system threats as well as to determine the risk associated with each of the identified threats. A performed TARA results in the so-called *Highest Risk Potential Threats*, a group of threats/attacks with the highest severity and the highest likelihood to be carried out successfully. Next,

[1] EU ARTEMIS project Dependable Embedded Wireless Infrastructure (DEWI). For further details see www.dewi-project.eu.

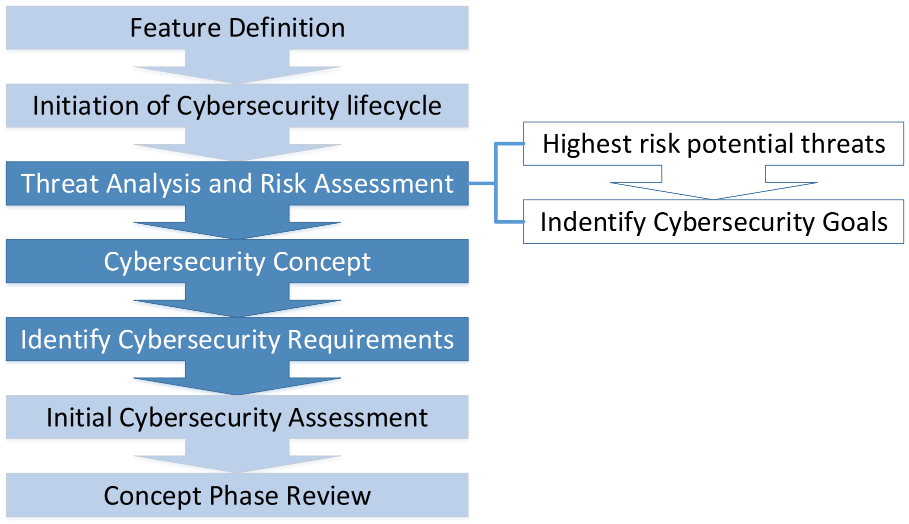

Feature Definition
Initiation of Cybersecurity lifecycle
Threat Analysis and Risk Assessment
Cybersecurity Concept
Identify Cybersecurity Requirements
Initial Cybersecurity Assessment
Concept Phase Review
Highest risk potential threats
Indentify Cybersecurity Goals

Figure 2.6: Flow of activities within the concept phase as defined in SAE J3061 [52].

these *Highest Risk Potential Threats* are used to determine the cyber-security goals and furthermore to develop a suitable cyber-security concept. Finally, based on the cyber-security goals and the corresponding concept, the functional (high-level) requirements can be extracted. These steps are then performed within an iterative process to further refine the results.

SAE J3061 provides an abstract, high-level description of the aforementioned steps, as well as the expected results. However, no guidance on how to obtain these results is given except a possible work flow described in the appendix of the standard. The DEWI Security Metric proposed in Section 6.1.1 of this thesis can close this gap and support the process of defining a secure system configuration starting from the identified cyber-security goals. Additionally, the DEWI Security metric provides a structured approach to create a comprehensive cyber-security concept as well as to extract cyber-security requirements.

2.5 SHIELD Multi-Metrics

The SHIELD Multi-Metrics approach was developed within the two EU projects called nSHIELD and pSHIELD. These projects were focused on the research of Security, Privacy, and Dependability (SPD) in the context of embedded and distributed systems. The SHIELD Multi-Metrics were designed to evaluate SPD aspects of an entire system as shown in [21], where a security concept for a vehicular communication unit was used as an example on how to apply the metrics. The proposed approach allows to find the best system configuration w.r.t. security by performing a structured system analysis. Therefore, the overall score of an entire system is computed by first dividing the system into subsystems, and finally into single components, by assigning weights and security values to each component, and finally by using the metrics to compute the score of each subsystem and for the entire system (i.e., a measurable security approach).

In Figure 2.7 the required steps of the SHIELD Multi-Metrics approach are outlined.

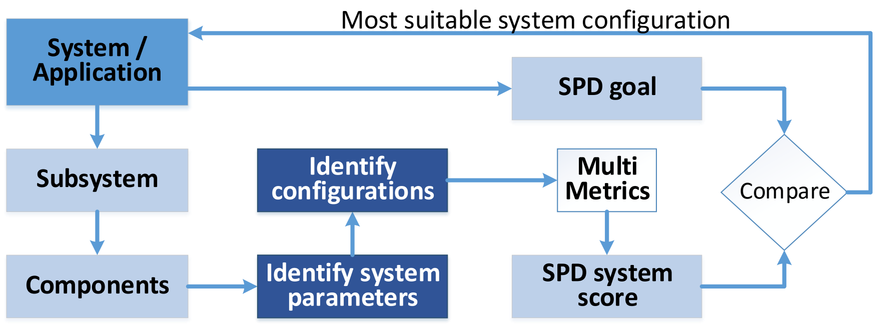


Figure 2.7: The SHIELD Multi-Metrics measurable security approach.

First, an SPD goal consisting of a value for security, privacy, and dependability in a range from 0 to 100 (e.g., security: from 0 for no security up to 100 for highest security level) is defined for each scenario (e.g., different local update scenarios). Next, the system is divided in subsystems and components. A metric consisting of a criticality and a weight value is defined for each system parameter (i.e., a possible technical solution for a component; e.g., a specific encryption scheme for a communication link). Thereafter, the Multi-Metrics are used to compute the SPD value of the entire system starting at component level (i.e., bottom-up approach). With these steps different predefined system configurations can be compared to each other, and the best fitting configuration can be identified. Please note that the configuration closest to the goal value will be chosen due to performance reasons (i.e., not necessarily the configuration with the highest score).

Chapter 3

Related Work and Research Challenges

In this chapter related work in the area of wireless automotive software updates will be reviewed. In particular, existing systems as well as proposed architectures will be discussed regarding their ability to provide a solution for LSUs in Section 3.1. Thereafter, existing wireless software update solutions are analyzed w.r.t. the important aspects security (see Section 3.2) and efficiency (discussed in Section 3.3). Next, in Section 3.4, the availability of comprehensive testbed infrastructures for locally-performed, secure and efficient wireless software updates is discussed. The chapter concludes by highlighting the need for an end-to-end solution for wireless automotive LSUs, and by stating the related research challenges in Section 3.5.

3.1 Existing Solutions for Automotive Wireless Software Updates

Wireless automotive software updates are currently in focus of academia as well as the industry due to their high potential and the multitude of related research challenges. The benefits of automotive OTA updates compared to wired software updates are listed in [51]. In this white paper the disadvantages of conventional (i.e., wired) software updates are listed, and a high-level architecture for efficient OTA updates is presented. However, the author provides no technical insights on how to realize such a system and neither a description of the wireless medium used nor the security mechanisms employed is given. In [47] the software update process for ECUs based on international standards (e.g., UDS and ODX; see also Section 2.2) is described. However, this work is not addressing a wireless approach for such updates at all.

To date (2018), Tesla is the only car manufacturer providing a solution for automotive remote updates for an entire fleet of vehicles. To perform such an OTA update, the vehicle is either interconnected with the OEM via a 3G Internet link, or connected to a Wi-Fi network to establish a secure tunnel to the OEM. The secured link between the OEM and the vehicle can then be utilized to transfer the latest software from the servers of the OEM to a Tesla vehicle [19]. However, this point-to-point connection between the OEM and the vehicle cannot be used to simultaneously install software in different vehicles and on several ECUs in parallel. Thus, this solution is not applicable for LSUs (e.g., within a service center, where several vehicles are maintained at the same time). Furthermore, the solution is dedicated to Tesla vehicles and cannot be used for vehicles of other OEMs. Hence, a service center providing maintenance for different brands will require a dedicated wireless system for each brand.

Other authors of previous work such as [26, 36, 39, 41, 43, 44] mostly focus on automotive remote updates, but do not consider the peculiarities of LSU scenarios at all. A system allowing OTA updates was presented by Idrees et al. [41]. In this work a high-level OTA software update architecture is presented with focus on a secure software update protocol and on the use of hardware-based security features. However, no further information on the actual implementation of the system is provided nor is any detail about the performance of the solution given. In a white paper by Steurich et al. [6], an OTA architecture utilizing trust anchors to secure the actual software transfer process is proposed. The authors discuss the advantages of wireless software updates and highlight the need for strong security mechanisms. They argue that hardware modules – the so-called trust anchors – are essential to protect the connection between vehicle and the outside world as well as to establish a secure in-vehicle communication network. Although the authors provide technical insights on the security aspects, they do not further describe an actual implementation of the system. Furthermore, no information about the applicability of utilizing the proposed system in LSU scenarios is provided. Also the authors of [49] propose to use security hardware modules on the ECUs and the vehicle gateway to secure OTA updates by providing a secured link between the OEM and the vehicle. In the paper, the authors focus on the proposed security features and provide a high-level architecture for automotive remote updates. However, no details on the actual implementation are given and no evaluation results are presented (besides the performed overhead analysis). Another OTA software update architecture is proposed by Nilsson et al. [43, 44]. In the described architecture, a vehicle is employing an Internet link to receive new software from a portal server. In the paper important security aspects, especially data integrity and data confidentiality, are listed w.r.t. automotive remote updates. However, the authors neither describe the utilized wireless network nor provide any details on an actual implementation or the corresponding system evaluation, but mainly focus on security aspects. The authors of [55] propose an Android-based framework for aftermarket software upgrades. A vehicle communication interface is used to interconnect the vehicle and a web server hosting the new software via 3G. The interface runs Android and is connected to the vehicle via the OBD interface. The authors describe a proof-of-concept implementation and some basic evaluation results (e.g., running a performance test on the server). However, the authors are not discussing real-world update scenarios (such as LSUs) and are not addressing security at all. The authors of [46] also omitted security mechanisms and only focus on a fast software update protocol applicable for LSUs as well as remote updates. However, the authors do not discuss automotive updates in its entirely (i.e., covering all involved steps and entities), but only focus on the developed compression algorithm.

Lack of suitable update frameworks for LSUs. The aforementioned solutions and architectures are proposing different ways to update automotive software running on ECUs. However, authors mainly address remote updates, where the vehicle is connected to the server hosting the new software (most works assume the OEM to be the source of new software) via a dedicated Internet link. This link can be seen as a point-to-point connection between the OEM and a vehicle. Local update scenarios i) where several vehicles shall receive new software at the same time, ii) expert users are maintaining vehicles whiles updates are performed, and iii) where no direct communication channel between the OEM and the vehicle exists, are not addressed. Therefore, the proposed solutions are not applicable for the LSU scenarios targeted in this thesis and a suitable LSU framework needs to be designed and developed.

Such a framework for locally-performed automotive software updates shall be able to address the peculiarities of different LSU scenarios (e.g., provide mechanisms to handle updates for several vehicles in parallel or allow users to work on several vehicles at the same time), must employ a suitable security concept protecting all involved users and entities, as well as shall provide efficient software update mechanisms to keep the time required to perform an actual software update to a bare minimum.

The majority of the listed papers is focusing on the security-related aspects of the software update process. The following section will discuss the proposed security solutions in more detail, with particular focus on their applicability for securing automotive LSUs.

3.2 Securing Wireless Automotive Software Updates

Wireless software updates require a wireless interface interconnecting the vehicle with the outside world. Such an interface does not only allow software updates, but can also be used to run vehicle diagnostics (e.g., in a service center) or to let users interconnect their personal devices (e.g., a smartphone) with the multimedia system of the vehicle. However, such an interface can also be exploited to attack the vehicle and its in-vehicle communication system [15, 22, 63]. Therefore, several related works address the security issues and propose different security solutions to protect all involved entities of wireless automotive software updates as well as the exchanged data [6, 26, 39, 41, 43, 44, 49]. The presented works specifically highlight the security aspects *ehicle integrity and authentication* [35, 41], *data confidentiality* [6, 41], *data integrity* [39, 43], and *key management and exchange* [6, 41, 49]. The core security features proposed to address all these security aspects are stated next.

Hardware-based security features. Different authors have proposed to use dedicated hardware-based security features to protect the software update process. In [41] the proposed system utilizes a Hardware Security Module (HSM) for security-related tasks such as ensuring data integrity as well as confidentiality and key management. The HSM is used on all ECUs as well as the wireless interface connecting the vehicle with the outside world. The authors mainly focus on the protection of the in-vehicle communication system and do not detail the keying infrastructure, the trust model used between the vehicle and the OEM (or the DT in local environments), as well as the wireless technology used to transfer data from the OEM to the interface of the vehicle. Steurich et al. [6] propose the use of secure elements to safeguard remote updates. In this paper, the authors state different security measures provided by secure elements, but do not specify a comprehensive security concept for wireless automotive software updates. In [49], the authors propose to use HSMs on the wireless vehicle interface as well as on all ECUs of the vehicle. The focus of the authors is on the integration of these modules in the vehicle and its ECUs, and an evaluation of the resulting resource overhead is presented. However, the authors do not discuss the entire software update framework (neither for remote updates nor for LSU scenarios), but only focus on securing the vehicle.

Software-based security features. Nilsson et al. [43] propose the use of a hash-chain to protect the integrity of the transferred data. Therefore, the software is divided into segments and computing the hash value of each segment. When a segment is transferred to the vehicle, the segment itself is transferred along with the hash of the next segment. Therefore, the vehicle can check that a segment was not altered while transferred. Additionally, the proposed protocol provide mechanisms to ensure data confidentiality and data authentication. Thereby, the authors only focus on unicast data streams between one vehicle and the portal server. Scenarios including several vehicles receiving the same software simultaneously (i.e., parallel updates) are not discussed.

In [39], the authors propose a security architecture for wireless automotive software updates in which data integrity is ensured by sending multiple copies of the software to the vehicle. However, the authors only discuss point-to-point links between the OEM and the vehicle. Furthermore, no update authentication and authorization step is discussed in the proposed architecture (i.e., update is started if the received copies match). In [26] the authors also present a security concept for OTA updates, where a software binary is transferred to a vehicle twice to secure the update process (this mechanism shall allow to detect changes in the transferred data set). The paper focuses on comparing the error probability of a system, where new software is only transferred once, with their approach where each software binary is transferred twice. However, the authors do not consider a wide range of state-of-the-art attacks (e.g., man-in-the-middle or replay attacks).

In [35], Liu et al. describe a vehicle authentication protocol to securely interconnect electric vehicles with a smart grid using the vehicle's wireless interface. Therefore, Public Key Infrastructure (PKI) is used to perform the authentication step between the vehicles and the grid in a secure way. The proposed protocol is not dedicated to wireless software updates. However, the underlying idea could potentially also be used (with slight adaptions) to perform the authentication step in LSU scenarios.

Security challenges with regard to LSUs. All listed security solutions highlight the importance of security w.r.t. wireless automotive software updates. A framework for wireless software updates – both remote as well as locally-performed updates – must address the essential aspects *vehicle integrity and authentication*, *data confidentiality*, *data integrity*, and *key management and exchange*. However, none of the proposed solutions is able to cover all these security aspects and the authors do neither consider the peculiarities of LSU scenarios in their security concepts nor evaluate the impact of the employed security mechanisms on the efficiency of the software update process (i.e., mainly w.r.t. the resulting software update duration).

A security concept for locally-performed software updates i) must address all aforementioned security aspects as well as ii) shall be suitable for different LSU scenarios and hence needs to configurable to fit the needs of each of these scenarios while still allowing an efficient (i.e., fast) software update process.

3.3 Increasing the Efficiency of Wireless Software Updates

Besides security, also the duration of a software update is important. Next, related works with a particular focus on increasing the efficiency of wireless automotive software updates – for both remotely as well as locally-performed updates – are discussed.

Efficient automotive software update protocols. Today, automotive software updates are often quite time consuming, as the in-vehicle communication networks (especially CAN) are rather slow. A single update of a 4 MB software binary can take up to 5 minutes [6]. Therefore, advanced software update mechanisms that allow to reduce the actual software update duration by means of partial updates (i.e., by reducing the number of bytes to transfer) or to perform updates for several vehicles at the same time by means of parallel updates could be very beneficial in all automotive software update scenarios.

In [6] an update mechanism for ECUs with two dedicated flash memory blocks is proposed. While block A is still executing the current software, new software is installed on block B. Once the software installation on block B has completed, the ECU can reboot and start the new software stored on block B. This mechanism hence allows to perform a software update while the vehicle is still operational (i.e., one can use and drive it). Although this block swapping (i.e., the use of a second block once the new software is installed) can be useful in some LSU scenarios (e.g., the mechanic can run diagnostics on the running vehicle while updates are performed in parallel), it is not actually decreasing the software update duration.

Andrade et al. [3] discuss parallel OTA updates for connected vehicles and focus on the impact of the mobile network used to transfer the data to the vehicle. The authors argue that performing updates for vehicles in parallel can help to significantly reduce the overall duration of the updates (i.e., instead of performing a software update for several vehicles sequentially, one parallel update will install the new software on all vehicles simultaneously). Although the authors focus on remote OTA updates, the basic idea of performing updates in parallel can also be transferred to LSUs.

In [27, 28] the authors propose an infrastructure-based wireless multicasting method to update the software installed on automotive ECUs. The multicast performed between vehicles and either a base station of a mobile communication network or an *Intelligent Transportation Tower* of future Intelligent Transportation System (ITS) infrastructure is used to distribute the software packets to the target vehicles. The proposed system also includes a security architecture to protect the multicast system accordingly. The proposed architecture is dedicated to remote software distribution (i.e., focus on how to transfer new software the last mile from suitable ITS infrastructure to the vehicle) and hence does not cover all aspects, required steps, and involved entities. However, the gathered simulation results show the benefit of performing software updates on several vehicles simultaneously.

To actually decrease the duration of an individual software update, the authors of [46] propose an update protocol that employs a compression algorithm to reduce the amount of data to transfer and thus speed up the update process. The compression algorithm allows to create a delta file that can be used on the gateway or on the ECU itself to create the new software version using the delta file in combination with the currently installed software. The delta files are significantly smaller than the software binary itself, thus the time to transfer the software to the ECU (especially via the slow CAN bus) is reduced significantly. However, this mechanism must be supported by the ECU (its bootloader) and hence it cannot be used for all ECUs.

A similar approach to reduce the amount of data to transfer to the ECU is proposed in [51]. The authors argue that the use of delta files can decrease the software update duration significantly, as the size of the delta file is typically less than 5% of the original size of the software binary and, therfore, the time to transfer the software to the ECU is also reduced. However, the authors are not providing any insights on how such a delta file can be created, nor on which algorithm they are using. Furthermore, no information is given whether the ECU needs to support the proposed delta-update mode.

Efficient software updates for Wireless Sensor Networks (WSN). In WSNs different mechanisms enabling efficient ways to update the software on the WSN nodes were developed. These mechanisms mostly address the aspects *disseminating the update*, *reducing the traffic* required for the disseminated updates, and the *sensor node execution environment* [7]. Especially *traffic reduction* is important w.r.t. wireless automotive software updates, as it mitigates the effect of the slow in-vehicle communication protocols [6] on the update duration and hence increases the efficiency of the entire process. In WSNs traffic reduction is often achieved by utilizing partial software updates, where only the delta between two software versions is sent to the sensor nodes, as shown in [10]. EASE-UP will make use of partial software updates similar to these mechanisms (see Section 5.3.2). However, in contrast to WSNs, where power consumption is of highest importance, the framework proposed in this thesis focuses on efficiency (w.r.t. the update duration) and addresses peculiarities of the automotive domain.

Lack of software update systems allowing efficient LSUs. Different solutions for automotive remote OTA updates as well as for wireless software updates in other domains show the potential of significantly increasing the efficiency of wireless software updates by utilizing advanced (e.g., partial or parallel) software update mechanisms. However, none of the discussed solutions are addressing automotive LSU scenarios.

A framework for LSUs shall be able to utilize advanced software updates mechanisms such as partial and parallel software updates to decrease the overall duration of updating the software on one or several vehicles. These mechanisms must be applicable for different LSU scenarios (e.g., a vehicle assembly line as well as a service center) and shall be as OEM- and ECU-independent as possible. Furthermore, the advanced update mechanisms must be protected by suitable security solutions (e.g., a secure multicast data stream transferring a software binary to several vehicles at the same time) to ensure a fast but also secure software update process.

3.4 Evaluating LSU Frameworks Experimentally

The LSU procedure for automotive systems is rather complex, and involves multiple steps ranging from the *secure* authentication with the device providing the new software image (i.e., the DT) and the *reliable* wireless data transfer, to the installation and verification of the new software on the target ECU employing an automotive bus. All these steps are interconnected and affect the overall *efficiency* of the software update process, which should always be evaluated or studied in its entirety.

Hence, the efficiency of a LSU system in terms of *duration* of an update performed on one or several vehicles and ECUs is of high relevance and needs to be carefully studied [58]. This requires a deep investigation of the main aspects affecting the efficiency of a software update system, such as: i) the topology of the *wireless network* , ii) the applied *security configuration* (including different authentication schemes and key lengths), and iii) the employed *software update mechanism* (e.g., parallel or partial updates). All aforementioned aspects shall be analyzed experimentally on real hardware – ideally all at the same time – in a repeatable as well as systematic way to study their interdependency and to show the applicability of the tested automotive software update framework at hand.

A number of automotive testbed infrastructures have been proposed to evaluate automotive systems. In [14], a testbed consisting of several automotive ECUs interconnected by CAN is used to verify the SW running on these ECUs. Although the testbed provides a basic software update function, it is not capable of evaluating the entire wireless software update process, nor is able to analyze security- or network-related aspects. In [64], a Vehicle-to-Vehicle (V2V) testbed consisting of 200 nodes distributed over three floors within an office building is presented. The testbed can be used to simulate different V2V scenarios (e.g., on a highway), thereby evaluating high density scenarios and properties of IEEE 802.11p such as the signal-to-noise ratio and the packet error rate. This testbed, however, does not allow to connect automotive ECUs and can not be used to evaluate any aspect with respect to the efficiency of wireless software updates. In [8] an open testbed integrating ad-hoc V2V communications and a wireless mesh backhaul deployed at the UCLA campus is introduced. Although the testbed is meant to be used for analyzing V2V communication aspects, the authors only evaluate a video streaming application employing the mesh infrastructure.

Lack of suitable automotive testbeds. Although a number of existing testbed infrastructures for automotive systems exists (e.g., to verify the software running on ECUs interconnected by CAN [14], or to study the reliability of car-to-car communications within a large-scale setup [64]), none of them is able to evaluate automotive wireless LSU systems in their entirety. Thus, there is a clear need for suitable testbed infrastructures allowing the evaluation of automotive LSU systems. Such testbeds must be able to emulate different LSU scenarios, to analyze the impact of the employed security features, and also to evaluate advanced software update mechanisms such as partial or parallel updates. Furthermore, suitable infrastructures shall also allow to extensively test different LSU scenarios end-to-end including the software distribution as well as the local software installation step.

Other works evaluate the proposed software update architectures/protocols for remote updates using simulations: in [26] an analytical and simulation-based analysis is performed to evaluate the proposed architecture. The authors of [49] use simulations to evaluate the memory overhead caused by the employed security mechanisms, and in [27] a dedicated simulator was developed to evaluate the proposed multicast software update architecture. Although these simulations can be used to analyze a software update framework (or at least single aspects of it), they cannot replace extensive tests performed on real automotive hardware.

3.5 Secure End-to-end Solution for Automotive LSUs

Numerous related works have been reviewed in Sections 3.1 to 3.4. These works are covering several important aspects of wireless automotive software updates and provide different solutions and architectures for secure software updates especially w.r.t. to remote OTA updates. Some of the reviewed works even provide *end-to-end solutions for automotive remote updates* [27, 51] or propose *end-to-end security concepts* [6, 39, 41] for such OTA updates. However, none of the discussed works is able to provide a secure end-to-end solution for automotive LSUs due to the focus on systems for automotive remote updates.

Lack of end-to-end solutions for LSUs. A secure end-to-end system for different LSU scenarios must provide suitable solutions for the software distribution step (i.e., software is distributed to local software updates providers such as a vehicle assembly line or a service center) as well as the actual software installation step within a local software provider. Such an end-to-end solution for LSUs shall be able to cover:

- **secure software distribution:** distribute the software from the OEM or a supplier company (i.e., the software creators) to a local software update provider such as a vehicle assembly line or a service center and provide suitable security features to protect the transferred data as well as the involved users;
- **efficient automotive LSU framework:** provide suitable mechanism to speed-up the software update process in LSU scenarios;
- **end-to-end security concept:** a comprehensive security concept is required to protect the integrity, authenticity as well as confidentiality of the transferred software, ensure the secrecy of the employed keys, protect the safety as well as the privacy of the involved users, and secure all involved entities.

The developed end-to-end solution shall be extensively tested in a suitable **testbed infrastructure**. This infrastructure shall allow to analyze the impact of the employed security mechanisms, the utilized wireless network, as well as the used software update mode on the software update duration.

Chapter 4

End-to-end Solution for Local Software Updates

A primary goal of this thesis is to design an end-to-end solution covering both steps of the LSU procedure as illustrated in Figure 4.1: the software distribution (step 1), and the local software installation (step 2). In this chapter a detailed description of the proposed architecture for the software distribution as well as for the actual software update process within a local update provider (e.g., a service center) is presented.

In Section 4.1, the considered LSU scenarios are discussed and their peculiarities are stated. Thereafter, in Section 4.2, requirements regarding an efficient and secure end-to-end solution for wireless automotive LSUs are stated and described. These requirements are valid for the secure software distribution architecture described in Section 4.4, as well as for the architecture for local software updates proposed in Section 4.5.

4.1 Local Software Update Scenarios

This thesis focuses on locally-performed wireless automotive software updates supporting the entire lifecycle of modern vehicles. In this section, the considered LSU scenarios, as shown in Figure 4.2, are described one by one. Each scenario represents a local software update provider able to install new software (once received from the OEM via the software distribution network) on a vehicle and its ECUs, respectively. In the following, the peculiarities of different LSU scenarios are revealed and the experience level of the involved users is discussed [61].

Vehicle development. In the vehicle development scenario, development engineers have to update the software of an ECU several times to evaluate and test newly developed features. A flexible and efficient software update framework supporting the development engineers in their work could be very beneficial (e.g., speeding up the preparation process as the engineer can update the software of several test vehicles at the same time). Vehicle development activities will primarily take place in restricted environments (e.g., a dedicated test track or the company premises) and will be performed by expert users.

Vehicle assembly line. Vehicle assembly is performed in a highly automated environment, where most working steps are performed by robots. An ECU assembled in a new

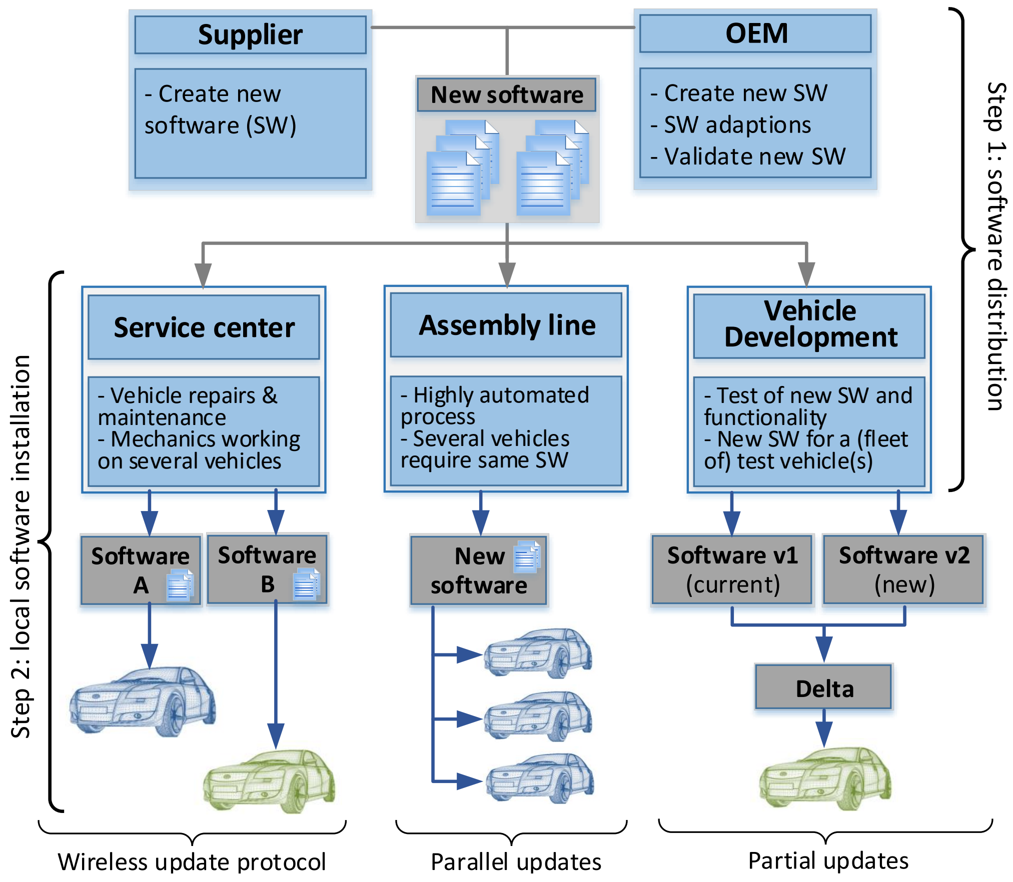

Supplier
- Create new software (SW)
New software
OEM
- Create new SW
- SW adaptions
- Validate new SW
Step 1: software distribution
Service center
- Vehicle repairs & maintenance
- Mechanics working on several vehicles
Assembly line
- Highly automated process
- Several vehicles require same SW
Vehicle Development
- Test of new SW and functionality
- New SW for a (fleet of) test vehicle(s)
Step 2: local software installation
Software A
Software B
New software
Software v1 (current)
Software v2 (new)
Delta
Wireless update protocol
Parallel updates
Partial updates

Figure 4.1: Architecture for an efficient end-to-end solution for wireless automotive LSU.

vehicle, in most cases, already holds the required embedded software. However, it can happen that the ECU software changes (e.g., due to a necessary bug fix) while the ECU with the installed (and faulty) software on it, is already shipped to the assembly line. In such a case, the software of many vehicles must be updated – ideally in parallel – to install the latest software on the concerned ECUs. Because of the high number of vehicles as well as the high degree of automation scalability, reliability and efficiency aspects of the utilized software update system are of essential importance. An assembly line is a secured environment where access is limited to trained users.

Service center. In a typical service center scenario mechanics (i.e., trained users) will diagnose, repair and maintain several vehicles. Therefore, a mechanic most likely first connects to a vehicle to run dedicated diagnostic functions and to look for DTC. Next, the mechanic will perform maintenance tasks as well as necessary repairs according to the gathered information. If new software is available for one or more of the embedded ECUs of the vehicle, the mechanic will additionally trigger the installation of the latest software.

For an efficient work flow, the mechanic will potentially work on several vehicles at the same time: while the latest software is installed on one vehicle, the mechanic can already start to diagnose an other vehicle. In case of large vehicle recalls, several vehicles can even require the same software update. Parallel software updates would significantly reduce

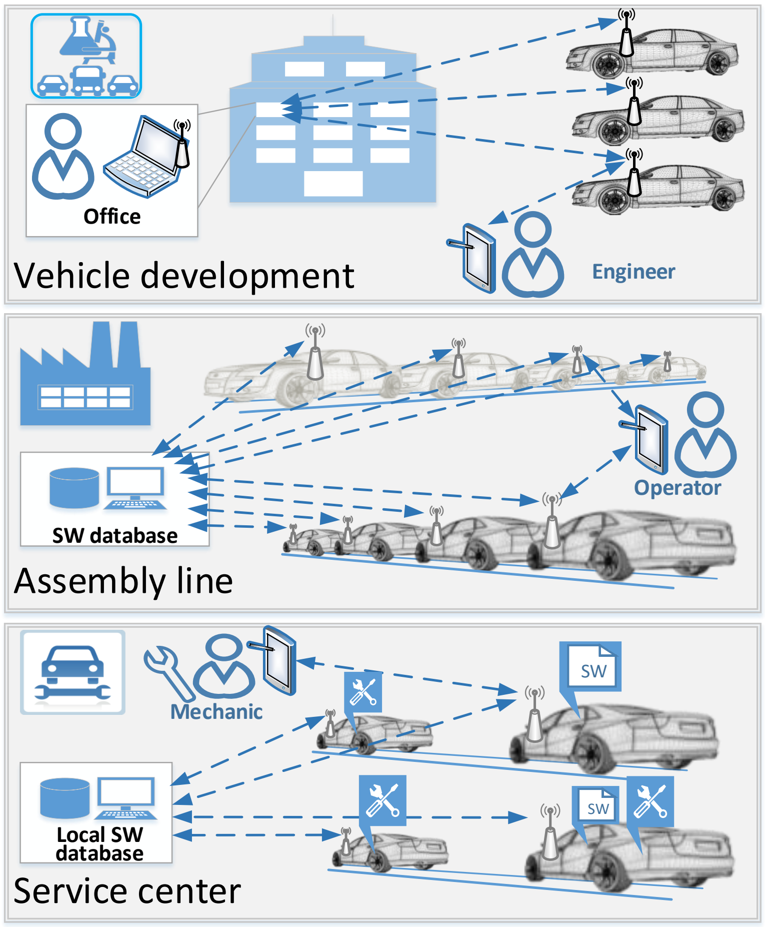


Figure 4.2: Considered LSU scenarios: wireless software updates performed in the *vehicle development* phase, in the vehicle *assembly line*, as well as in *service center* scenarios.

the overall update duration in such cases.

A service center is basically a restricted area. However, customers will frequently enter the facility to pick up their repaired vehicles. Thus, service centers cannot be considered as secured environments with limited access in practice.

4.2 Requirements for an End-to-end Solution for LSUs

An end-to-end solution for wireless software updates suitable for different LSU scenarios shall be able to fulfill a range of essential system-level requirements. The latter must be addressed by the developed software distribution architecture as well as by EASE-UP, the developed framework allowing efficient as well as secure local software updates. In the following, the relevant aspects efficiency, reliability, security, and privacy are briefly discussed and their scope within an end-to-end solution for LSUs is defined. Thereafter, specific requirements for both the secure software distribution (step 1) and the local software installation (step 2; requirements for EASE-UP) are stated. In Section 5.1.1, the list of relevant aspects will be used again to extract specific requirements on a wireless networking technology employed in LSUs.

- *Efficiency:* the end-to-end solution shall be as efficient as possible, whereas efficiency is mainly seen with regard to the overall software update duration. Each component and subsystem shall provide suitable bandwidth in the employed communication channels as well as make use of parallel data streams where possible to reduce the overall software update duration to a bare minimum.

- *Reliability:* the developed system and its components shall be reliable to ensure that new software is installed successfully without any system failure or unnecessary delays (reliability also contributes to efficiency). The system shall avoid single point of failures (especially with respect to the wireless network) and try to benefit from redundancy where possible.

- *Security:* all involved components as well as users must be protected by suitable security measures. These measures must i) protect the confidentiality as well as the integrity (i.e., against unintended changes such as flipped bits on the communication channel as well as intended changes by an attacker) of the exchanged data, ii) ensure that only authorized users can interact with the developed update system, iii) protect vehicles from unauthorized updates and access from outside in general, and iv) provide suitable mechanisms to prove the authenticity of the involved entities (e.g., a trustworthy DT is connected to the vehicle) as well as the exchanged data (e.g., the authenticity that a new software is really coming from the OEM). The employed security mechanisms must be strong enough to fulfill the aforementioned security aspects i) to iv), but shall also be as light-weight as possible to keep the added latency to a minimum (i.e., contributing to efficiency).

- *Privacy:* personal data (e.g., location information) of the involved users (especially the owners of the vehicles) shall be protected to ensure the user's privacy. Privacy is thereby closely connected to the security requirements as ensuring the confidentiality of the exchanged data is essential to privacy. However, privacy also requires protection from malicious insiders such as a rogue mechanic using service center equipment to extract personal information from a vehicle.

4.2.1 Requirements for Secure Software Distribution

In the following, the aforementioned general requirements are reconsidered w.r.t. the secure software distribution system. These requirements will then be used to design the corresponding architecture described in Section 4.4.1.

- *Efficiency:* the software distribution shall be efficient and hence fast. This is especially important if a critical software bug was discovered and the fixed software version shall be distributed as fast as possible. The efficiency aspect is also related to scalability, as the distribution shall be efficient on an entire fleet of vehicles encompassing thousands of vehicles located around the globe.

- *Reliability:* the used software distribution network shall provide multiple paths interconnecting the entities creating new software such as the OEM, and the local software update providers such as a vehicle assembly line, where the new software is installed on the concerned ECUs. This communication layer redundancy shall help to prevent single point of failures in the software distribution process.

- *Security:* a software update can potentially allow an upgrade of a certain function/system within the vehicle and users (i.e., the vehicle owner) may have to pay for such an update. Therefore, the confidentiality of the transferred software must be protected so that i) no one can copy the software and provide it to others for free or sell it without permission, and ii) the intellectual property of the software w.r.t. the OEM and/or the supplier company is ensured.

  The software distribution architecture also needs to protect the integrity of the transferred data (i.e., the software binary) all the way from the department of a company creating the software until it is finally installed on the concerned ECUs. Neither an attacker nor a malicious employee shall be allowed to tamper with the data while the software is stored and/or distributed.

  Additionally, the designed software distribution architecture must ensure that i) only authorized vehicles shall be allowed to get the new software binary, and ii) the OEM is really the source of the new software.

- *Privacy:* the privacy of the involved users (i.e., the vehicle owner and driver) must be protected. In contrast to current remote update systems (e.g., as used by Tesla [19]), where dedicated VPN tunnels are established to perform software updates via point-to-point connections between the vehicles and the OEM, the designed secure software distribution architecture shall allow a connectionless software distribution procedure. This loose coupling between the involved entities (i.e., OEMs, automotive suppliers, local software update providers such as service centers and vehicle assembly lines, and vehicles) shall be able to mitigate the risk of insider attacks, where an attacker (e.g., a malicious employee of an OEM) exploits the communication channel for accessing privacy-related data from the vehicle or tracking the location of the vehicle.

In Section 4.4 a Blockchain-based solution for secure and efficient software distribution addressing all these requirements and aspects is proposed.

4.2.2 Requirements for EASE-UP

In the following, the list of general requirements discussed in Section 4.2 is reconsidered w.r.t. EASE-UP, the developed framework allowing efficient and secure wireless software updates in LSU scenarios. These requirements will then be used to develop an efficient update protocol encompassing advanced update mechanisms such as partial and parallel software updates (see Chapter 5), as well as to define the corresponding security concept applicable for different LSU scenarios (see Chapter 6).

- *Efficiency:* EASE-UP shall reduce the time required to install new software on an automotive ECU in a local environment to a bare minimum by i) allowing users to work on several vehicles at the same time using one handheld device, by ii) employing beneficial mechanisms such as partial or parallel software updates, and by iii) utilizing a fast wireless network interconnecting all involved entities.

- *Reliability:* EASE-UP must ensure that a software update was successfully finished, as a failed update (i.e., update started but never finished) will set the concerned ECU and potentially the entire vehicle into an undefined and safety-critical state. Therefore, EASE-UP shall provide suitable mechanisms to inform users about the current state of an update and, in case an update has failed, also provide information about the reason of the update failure.

- *Security:* EASE-UP must be protected by a suitable security concept ensuring data confidentiality and integrity, and protecting involved entities (e.g., the DT) and users (e.g., mechanics in a service center). Thereby, the security concept shall be generic to be applicable for all targeted LSU scenarios and also be configurable to fit the needs of each of these scenarios. Please note that a measurable security design approach (see Section 6.1 for more information) was used to extract detailed security requirements. These requirements can be found in Section 6.2.1.

- *Privacy:* EASE-UP must prevent attackers (i.e., hackers as well as rogue internal users) from extracting personal information stored on a vehicle.

4.3 End-to-End Software Update Procedure

The requirements stated and described in the previous section (i.e., Section 4.2) apply to both steps of the targeted end-to-end solution for locally-performed wireless automotive software updates, as shown in Figure 4.3.

In the first step, the *secure software distribution* architecture described in Section 4.4 is used to transfer new software from the suppliers and the OEMs to the local software update providers such as service centers or vehicle assembly lines. Once the software has reached the local software update providers, the software will be verified to make sure that it was originated by the OEM and not altered while transferred. After verifying the new software, the local software update providers will store the software in their local databases (i.e., part of the DT).

The actual wireless *locally-performed software update* is carried out once a vehicle requiring the new software has entered the local software update provider and connected to the LSU system (i.e., EASE-UP). The latter will then check for available updates by querying its local database, initiate the software update process on the connected vehicle, and finally control the wireless LSU procedure. More information on EASE-UP can be found in Section 4.5, as well as in Chapters 5 and 6.

4.4 Secure Software Distribution

An automotive software update can be triggered by a tier-1 supplier (i.e., a company directly selling products, e.g., an ECU or its software to the vehicle manufacturer) or by an OEM due to a necessary bug fix or to upgrade the functionality of an ECU, as indicated in Figure 4.4. Before the update can be installed on the ECUs of the concerned vehicles,

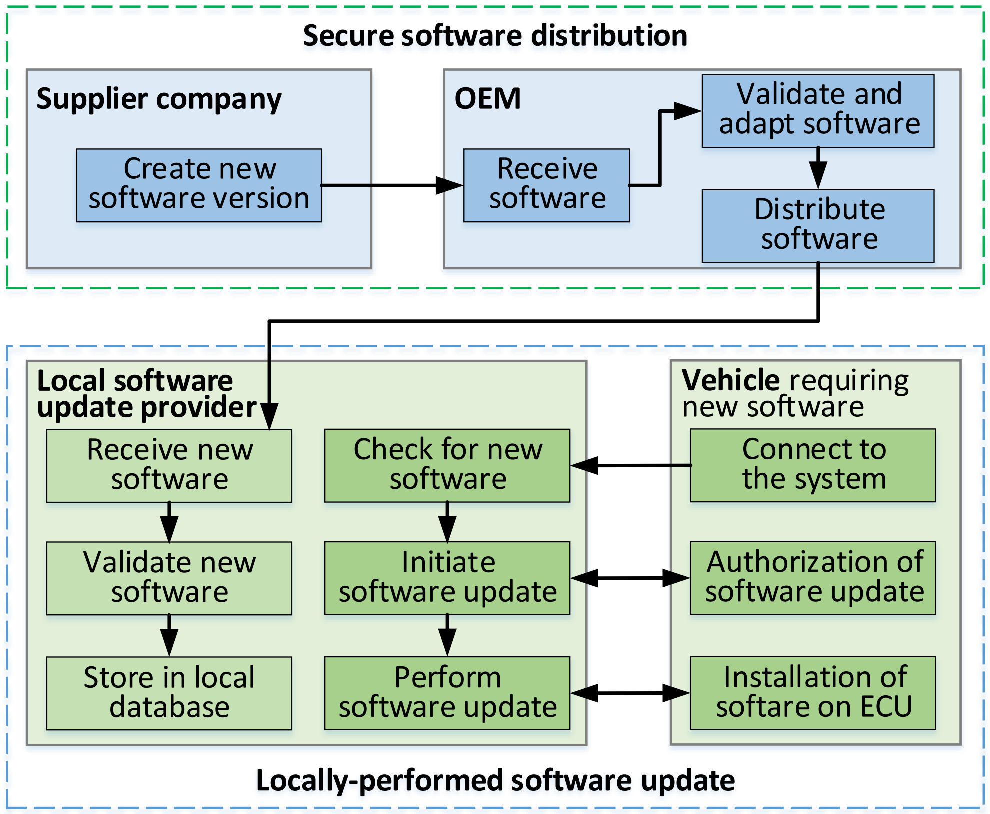

Secure software distribution
Supplier company
Create new software version
OEM
Receive software
Validate and adapt software
Distribute software
Local software update provider
Receive new software
Validate new software
Store in local database
Check for new software
Initiate software update
Perform software update
Vehicle requiring new software
Connect to the system
Authorization of software update
Installation of softare on ECU
Locally-performed software update

Figure 4.3: End-to-end software update procedure including the *secure software distribution* step as well as the *locally-performed wireless software update* process.

the new software first needs to be transferred to the vehicle itself or a local software update provider. This is done within the *software distribution step*.

As illustrated in Figure 4.4, the software distribution step is suitable for both automotive remote updates (out of scope of this thesis) and locally-performed software updates, and encompasses two main stages: in the first stage, a supplier company (after creating the new software) informs the OEM (or even several OEMs) about an available software update. Next, the OEM will validate the new software and potentially perform some adaptions. In the second stage, the OEM will forward the new software to the vehicles itself (i.e., remote updates; out of scope of this thesis) or to local software update providers such as the vehicle assembly line or a service center.

In the following, the proposed architecture for secure software distribution is described in detail. This architecture allows to cover both aforementioned stages (i.e., stage 1: transfer new software from the supplier to the OEM; as well as stage 2: distribute the software to local software update providers) of the software distribution step and provide suitable functions to:

- *notify entities about available software:* depending on the current stage of the software distribution process, either the OEM (i.e., in stage 1) or the local software providers as well as smart vehicles (i.e., in stage 2) are informed about new software,

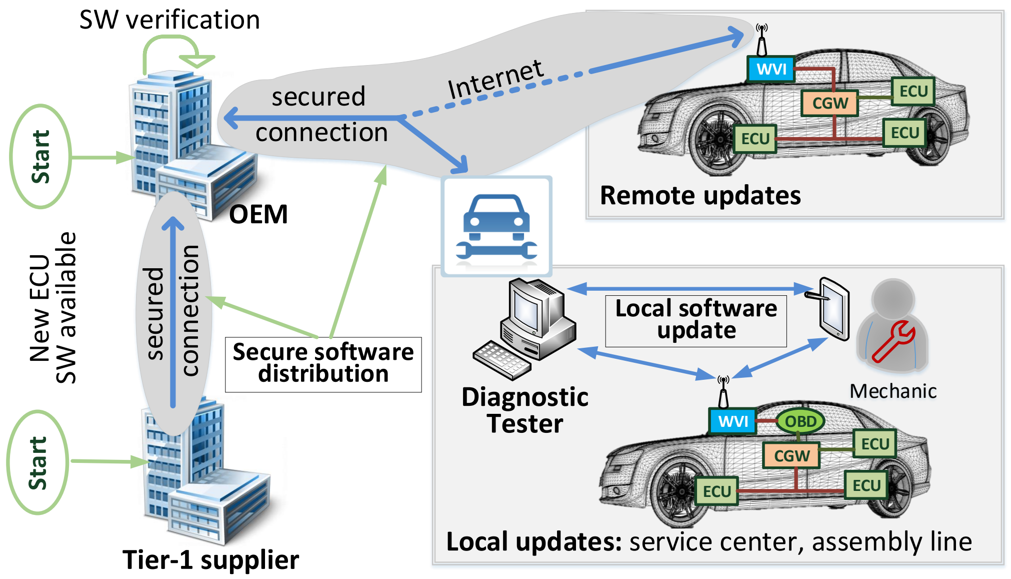


Figure 4.4: End-to-end solution for wireless software updates including secure software distribution as well as software installation on the ECU.

and a pointer (e.g., an Internet link) to this software is provided;

- *securely and efficiently transfer as well as store software:* provide suitable mechanisms i) to store new software on a public-accessible server or cloud service (i.e., an online storage), and ii) allowing authorized entities and users to upload as well as download software binaries from this online storage.

Additionally, the employed architecture shall be able to fulfill the list of requirements stated in the Section 4.2.1.

Please note that the designed secure software distribution architecture is applicable for remote as well as locally-performed automotive software updates. However, the focus of this thesis is clearly on the development of local software update procedures.

4.4.1 Designed Architecture for Secure Software Distribution

The aforementioned aspects as well as the requirements stated in Section 4.2.1 mostly match the benefits and advantages of a typical Blockchain architecture (i.e., security, immutability and privacy). Therefore, Blockchains could be a useful technology to realize a secure software distribution. More information on the functionality of the Blockchain technology can be found in Section 2.3.1. The proposed architecture is based on the Lightweight Scalable Blockchain (LSB) developed by Ali et al. [12]. LSB utilizes a hierarchical model consisting of clusters, where each cluster is maintained by a Cluster Head (CH) and encompasses several Cluster Members (CM). The so-called overlay network is used to interconnect the CHs. More information can be found in Section 2.3.2.

The designed Blockchain architecture covers three essential tasks of the software distribution process. First, it allows to notify interested entities (i.e., the OEM as well as the local software providers) about available software. Therefore, a notification message containing information about the location of the software on a online storage, is distributed within the overlay network. Second, the designed Blockchain infrastructure provides suitable authorization mechanisms handling the interaction between involved entities and the online storage (i.e., only authorized entities are allowed to access/download the software). Third, it allows to prove the authenticity of the entities providing new software as well as of the software itself (i.e., to ensure that no one has tampered with the software). In the following, the designed architecture is described in more detail.

Automotive security architecture based on Blockchain. The developed architecture is sketched in Figure 4.5. It involves several different entities, namely the software creator, the OEM, a cloud storage, local software providers such as a service center, as well as smart connected vehicles. The *software creator* is a dedicated department of an OEM or a supplier company creating the new software version. This new software binary is then stored on a *cloud storage*, a secure online storage only allowing authorized users to store and access data. Next, the department of the *OEM* responsible for verifying and adapting new software is notified about the new binary. Once the verification process is done, the *LSU environments* and *smart vehicles* (i.e., directly connected to the Blockchain network) are informed about the available software update.

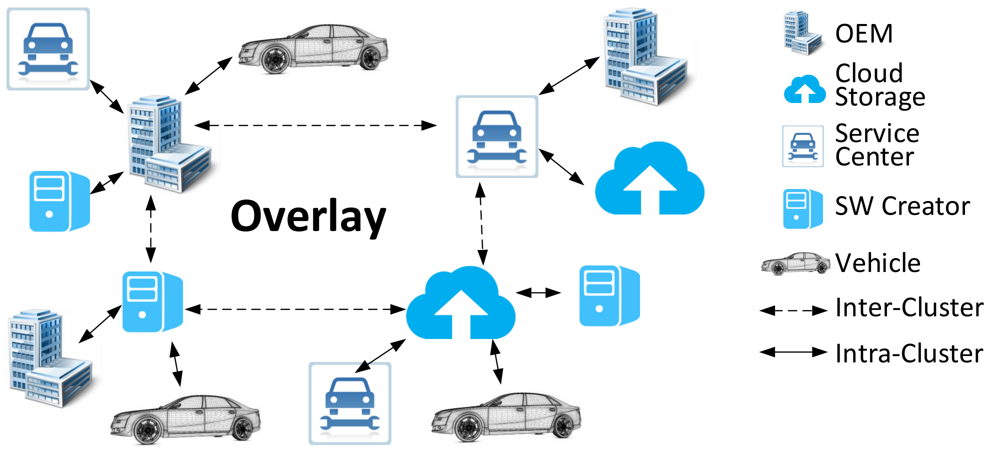


Figure 4.5: Blockchain-based security architecture. The overlay network interconnects the OEM, the software creator, the service centers, a cloud storage, and even smart vehicles.

The notification of the involved nodes is done using transactions either sent from the software creator to the OEM (step 1), or sent from the OEM to local software update providers such as a service center and to smart vehicles (step 2). The developed architecture employs two different types of transactions: a *genesis transaction*, the initial transaction created by each involved party, and the *update transaction*, required to inform the concerned entities (i.e., the OEM in step 1, and the LSU environments and smart vehicles in step 2) about newly available software.

A transaction basically consists of metadata, the public key of the transaction creator, the transaction ID (i.e., the SHA-256 hash value of the metadata and the public key), and a digital signature (i.e., the ID is signed using RSA with a 1024 bit key). This generic structure is also shown in Figure 4.6a). An update transaction, as presented in Figure

4.6b) additionally includes a second public key and signature field. These extra fields are required to cover both aforementioned steps: after creating a new software binary, the software creator will store the binary on a cloud storage, create a new transaction, fill the metadata field with information about the software (including a hash of the software to allow other parties to verify its integrity) and the location of the binary on the cloud storage. Furthermore, the software creator adds its own public key, creates the hash (i.e., transaction ID), signs the transaction using its private key, and finally sends the key to the overlay network. The OEM receives the incomplete transaction, verifies the signature, downloads the software binary, adapts and verifies the binary, and finally, if every check was ok, signs the transaction with its private key. The second signature completes the transaction. A genesis transaction is the first transaction created by each entity within a Blockchain system and its structure can really differ depending on the type of the entity (e.g., an OEM, a service center, or a smart vehicle). In Figure 4.6c), an example for a genesis transaction of a smart vehicle is proposed. It consists of metadata describing the vehicle itself (e.g., the type and variant of the vehicle), its public key and signature, as well as the public key and signature of its OEM. This allows the vehicle in a later phase to access vehicle-related data (e.g., a software binary) stored on a cloud storage.

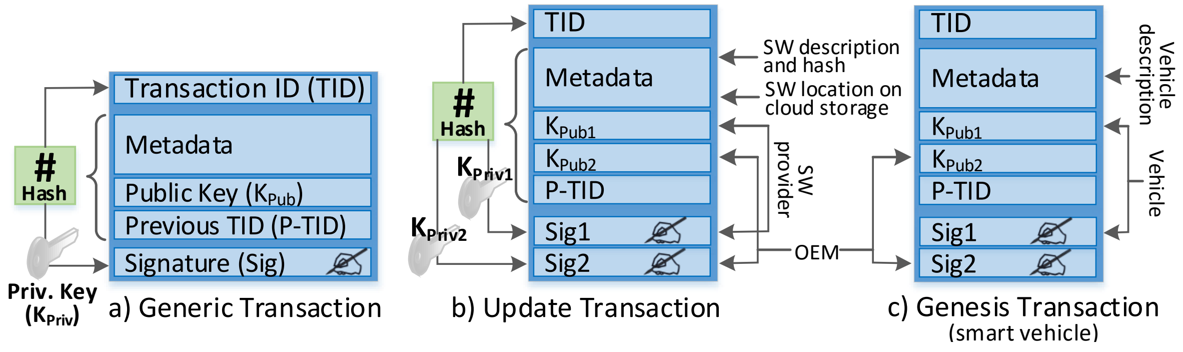


Figure 4.6: Different Blockchain transaction. Left: generic transaction structure. Middle: update transaction. Right: proposed structure of a genesis transaction for smart vehicles.

Example of software distribution. Next, an end-to-end scenario (shown in Figure 4.7) ranging from the creation of new software until its installation on the concerned ECU is described.

First, the software creator creates a new software binary (Figure 4.7, step 0) to fix a bug in the old software version, to upgrade the functionality of the ECU, or to adapt essential parameters required by the ECU. Once the software binary is completed and tested, it is stored on a cloud storage (Figure 4.7, step 1). Next, the software creator creates an update transaction, fills all required fields, and sends it to the concerned OEM (Figure 4.7, steps 2 to 4). If the software creator is a CH in the Blockchain architecture it directly uses the overlay network to deliver the transaction. In the other case, (i.e., the software creator is a CM), the transaction is first sent to the CH and then, after the CH has checked the transaction by verifying the signature, forwarded to the OEM. The forwarding mechanism makes use of the public key of the OEM, which was added to the update transaction by the software creator to indicate which OEM (especially important

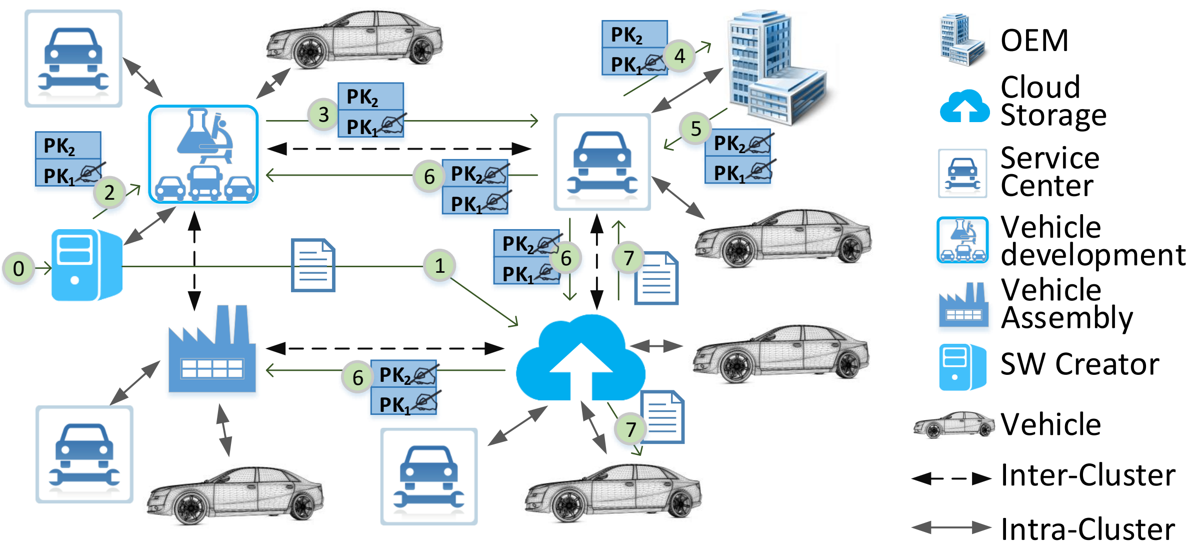


Figure 4.7: Software distribution example. From software creation at the software creator to software installation on the ECU.

if there are several OEMs within the same Blockchain overlay) shall be notified about the new software. Please note that the overlay network supports both the broadcasting of transactions and blocks (i.e., complete transaction and new blocks) as well as forwarding incomplete transactions (and other message types such as the data transfer to the cloud storage) to a specific CH or CM.

Once the transaction reaches the cluster including the OEM (i.e., the OEM is either acting as CH of the cluster or is a CM of the cluster), the OEM will receive the transaction, verify it, and download the software. Next, the software is validated as well as adapted (e.g., to make sure to fit the requirements of certain countries/continents). Thereafter, the OEM completes the transaction by adding its public key and signature. In the next step, the transaction is broadcast on the overlay (Figure 4.7, steps 5 and 6). As the transaction is complete, it is distributed to all CHs, who verify the received transaction and add it to their running pool. This is a temporary buffer where new transactions are stored until a predefined size (i.e., the block size) is reached. Once this limit is reached, the CH will create a new block and distribute it over the network.

After the received complete update transaction is verified by the CHs, it is forwarded to the CMs. However, the developed architecture is not broadcasting a transaction to all CMs within a cluster, but only to *interested* ones in order to reduce the communication and computation overhead. Interests are managed by using the public keys included in the transactions. A CM can subscribe to one or more public keys and thus show its interest in transactions from certain entities. In this concrete example, this means that a vehicle could subscribe for messages coming from a specific OEM (Figure 4.7, step 7).

4.4.2 Formal Architecture Evaluation

In this section the defined architecture is formally evaluated by first discussing the fulfillment of the aforementioned requirements, and second by briefly addressing the most relevant security threats and stating how the use of the Blockchain plus the employed security mechanisms can mitigate these threats.

Fulfillment of the defined requirements. The proposed architecture allowing a secure distribution of automotive software shall be able to fulfill the important requirements *efficiency*, *security*, and *privacy* described in Section 4.2.1. In the following, these requirements will be discussed w.r.t. the proposed Blockchain architecture and thereby its applicability for secure software distribution will be shown.

- *Efficiency:* The efficiency of the utilized Blockchain system is ensured w.r.t different levels: first, the employed LSB uses a distributed trust algorithm to significantly lower the processing time for validating new blocks compared to classical Blockchain schemes (e.g., used in the Bitcoin system) [12]. An extended version of the proposed architecture can potentially utilize suitable caching mechanisms within clusters to further reduce the data traffic on the overlay network and hence additionally increase the efficiency of the software distribution process.

  Second, the proposed architecture (underpinned by the LSB scheme) replaces the PoW by a more resource-optimized approach (i.e., scheduled block generation process). Additionally, the network is clustered to realize a hierarchical topology, where only the CHs need to maintain the Blockchain. Furthermore, LSB dynamically adjusts the throughput using a Distributed Throughput Management (DTM) to ensure that the Blockchain throughput does not significantly deviate from the transaction load generated by the nodes in the network [12]. Thus, LSB is optimized for large-scale networks and applicable for scenarios involving entire fleets of vehicles.

- *Reliability.* The employed overlay network is able to avoid single point of failures by using alternative routes in case a CH is temporarily offline and hence unable to forward messages.

- *Security.* The confidentiality of the transferred data (i.e., the distributed software) is ensured by i) storing and transferring the software in an encrypted way, and ii) by only allowing authorized vehicles as well as local software update providers to download and decrypt the software update.

  Data integrity is ensured by employing state-of-the-art security measures (i.e., AES-GCM [16]) to protect each packet. Additionally, the architecture allows each authorized entity (i.e., vehicles and local software update providers) to verify the integrity of the distributed software by downloading it from the cloud storage, computing its hash value, and comparing this hash with the target hash value included in the metadata field of the update transaction (see also Figure 4.6).

  The OEM as well as the software creator are required to sign an update transaction before it is considered as complete, stored within a block, and distributed by the overlay network. These signatures allow the vehicles to verify that an available software update is really coming from the OEM and not from any malicious entity. Furthermore, the hash value included in the update transaction allows the vehicle and the local software update provider to validate that the software was not altered during distribution or storage.

- *Privacy.* In Blockchain architectures the privacy of the involved users is ensured by utilizing changeable public keys representing the end user (i.e., the vehicle owner). Furthermore, and in contrast to other remote update methods such as the system used by Tesla [19], no direct link between the vehicle receiving a new update and the OEM providing this update is required. Such direct link could be used in a malicious way to endanger the vehicle owners privacy (e.g., by tracking the user).

The previous listing shows that all requirements are met by the designed secure software distribution architecture. In combination with EASE-UP, the developed framework allowing efficient as well as secure wireless software updates in LSU scenarios, the proposed architecture can be used to create a secure and efficient end-to-end solution for wireless automotive LSUs.

Most relevant security threats. In the following, selected attacks are used to discuss the security measures employed by the designed software distribution architecture and to show how these attacks are affecting the security of a smart vehicle and other involved entities. Please note that the used list of threats is not complete (as there is a wide range of possible attacks), but encompasses diverse attacks with high potential and impact.

Tampering with the software binary. An attacker may try to get access to the cloud storage and manipulate the software binary stored there with the goal of injecting malware to a large number of vehicles. However, changing the software will lead to a different software hash value compared to the hash of the untouched software included in the update transaction. Thus, involved vehicles as well as local software update providers will detect such an attack prior to installing the affected software update.

Additionally, an attacker could try to tamper with the data (i.e., the software binary) while it is transferred to render the software useless. However, adequate protection mechanisms are in place while the software is i) transferred to the cloud storage, and ii) finally distributed to the vehicles and local software update providers. First, each individual packet is protected to ensure data confidentiality and integrity by employing suitable symmetric security measures (i.e., AES-GCM). Second, after the software is completely transferred, the receiver (e.g., a local software provider or a smart vehicle) can compare the hash of the received software with the hash included in the update transaction.

Distributing a malicious update by claiming to be the OEM or the software update provider (i.e., identity spoofing). An attacker can also attempt to distribute malicious software updates via the overlay network by claiming to be the OEM and/or the software creator. To launch such an attempt, the attacker would need to broadcast an update transaction pointing to the malicious software using the overlay network. This transaction would need valid signatures from the software creator as well as the OEM, and hence the attack will fail, as i) in case the attacker acts as software creator, the OEM will verify the signature and realize that it is invalid (e.g., signature doesn't fit the public key of the software creator) or will just not accept the transaction from a unknown source (i.e., transaction includes the public key of the attacker which is unknown to the OEM), and as ii) in case an attacker is acting as OEM and manipulates an update transaction coming from a trustworthy software creator, the vehicles will realize that the update transaction was not signed by the real OEM by verifying the signature (a vehicle knows the public key of its OEM). The same security mechanisms will also mitigate attacks where an attacker spoofs both the identity of the software creator as well as of the OEM.

More attacks as well as the corresponding countermeasures are discussed in [13].

4.4.3 Proof-of-concept Implementation

The developed Blockchain-based security architecture was additionally evaluated using a proof-of-concept implementation based on Java. This proof-of-concept solution was then compared to a certificate-based software distribution system and important aspects such as packet overhead and overall efficiency were analyzed. The gathered results show that both systems are quite similar w.r.t. overhead and efficiency.

More details on the proof-of-concept implementation, the performed evaluations, as well as on the corresponding results can be found in [59].

4.5 EASE-UP: Framework for Secure and Efficient LSUs

Once a new software version was securely distributed to local software providers such as an assembly line or a service center, efficient and secure wireless software updates will be employed to install the new software on the concerned ECU of a newly assembled or currently maintained vehicle. This is achieved using EASE-UP, whose architecture allowing efficient and secure LSUs is presented next.

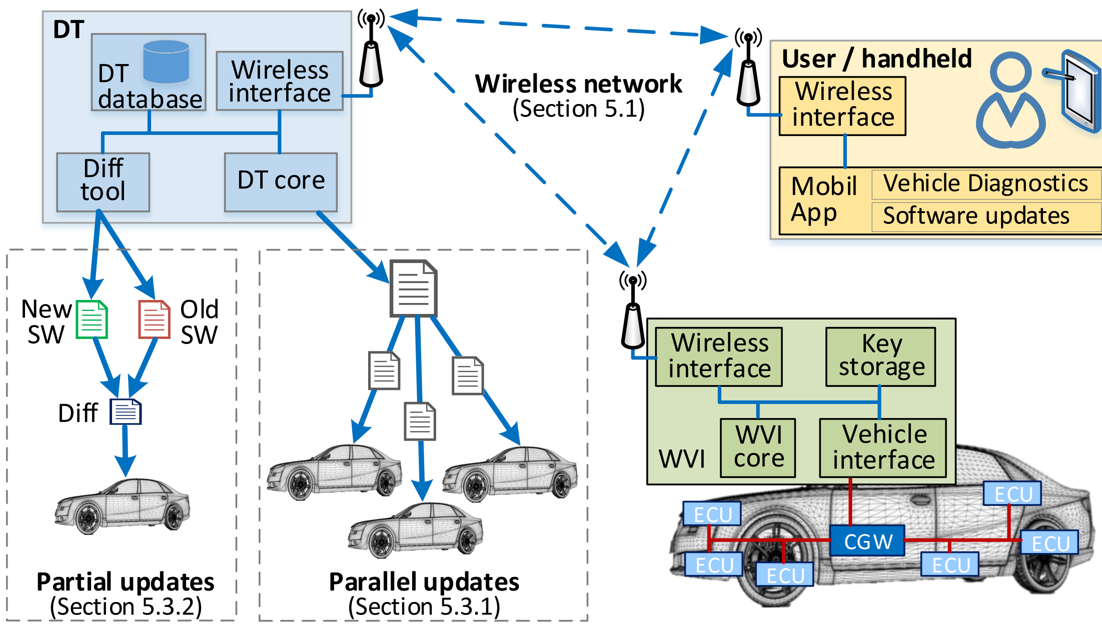


Figure 4.8: EASE-UP architecture involving the Diagnostic Tester (DT) holding the latest available software as well as the required authorization keys, the handheld employed by the user to interact with EASE-UP, and the Wireless Vehicle Interface (WVI) acting as smart gateway between the wireless network and the in-vehicle communication system. The in-vehicle communication system encompasses several different bus systems often interconnected by a Central Gateway (CGW).

4.5.1 Architecture

EASE-UP is a framework allowing locally-performed efficient and secure wireless automotive software updates. The corresponding architecture, as shown in Figure 4.8, provides a basic wireless software update protocol (see Section 5.2.1) similar to today's wired software update systems (i.e., the wired connections are replaced by a wireless network) and encompasses three core nodes, as described Section 4.5.2. These core nodes are interconnected by a fast and secure wireless network suitable for all targeted LSU scenarios. In Section 5.1 more details on the employed wireless network are presented.

In contrast to wired systems, where only one vehicle can be maintained at a time, EASE-UP allows a user to work on several vehicles simultaneously and provides advanced software update mechanisms to update the software on several vehicles at the same time (i.e., parallel updates; see Section 5.3.1), or to perform a partial update (see Section 5.3.2), where only a small portion of the software (i.e., the difference between the old and the new software) needs to be transferred and installed on the ECU. Both mechanisms allow to significantly reduce the time required to perform a LSU on one or several vehicles.

EASE-UP is protected by a comprehensive security concept applicable for different LSU scenarios. This security concept is described in detail in Section 6.2.

4.5.2 Core Nodes

The architecture of EASE-UP, as shown in Figure 4.8, encompasses three core nodes (i.e., DT, WVI, and the handheld device) and is suitable for all aforementioned LSU scenarios. In the following, these core nodes are described in more detail and the architecture for local wireless software updates is highlighted. Further information on the actually developed prototypes can be found in [37, 61].

Diagnotic Tester (DT). The DT is a local server or PC often located in a dedicated and restricted room. The DT maintains a local database that is frequently synchronized with the OEM using a dedicated backbone network (e.g., the Blockchain-based software distribution architecture; see Section 4.4).

The local DT database holds vehicle descriptions including types and variants of vehicles of a specific brand and the corresponding vehicle configuration. The vehicle configuration is typically stored in the Open Diagnostic data eXchange (ODX) format [32] and encompasses, beside others, information about the integrated ECUs including ECU-IDs, as well as the latest available software version for each integrated ECU. Furthermore, the DT can provide authentication keys required to authorize a software update on ECU level and holds the latest available software binaries (or can request the software using the backbone network). The DT can therefore be seen as the source for new software updates.

Depending on the scenario, the DT can additionally be used to maintain user profiles: in a service center, the user profiles can be utilized to specify which mechanics are authorized to perform software updates (e.g., specially trained staff) and which users are only allowed to perform basic diagnostics when performing repairs or yearly inspections.

Handheld device. The handheld device allows a user to interact with the wireless software update system. Depending on the scenario, the handheld is used to perform wireless diagnostics, to trigger and/or schedule wireless software updates, as well as to monitor the update process itself (e.g., in the service center or vehicle development), or it is mainly utilized to monitor the software update process and the state of the update system (e.g., within the assembly line).

Wireless Vehicle Interface (WVI). The WVI is an essential entity as it interconnects the wireless software update system with the in-vehicle communication system. As sketched in Figure 4.8, a modern vehicle can contain dozens ECUs interconnected by different automotive bus systems (mainly CAN, but also LIN and FlexRay, as well as, in future, automotive Ethernet). In most vehicle architectures, the different bus systems are interconnected by a central gateway (CGW). This gateway is used to forward data from one bus to the other, and can also be used to implement certain security mechanisms (especially in future vehicle architectures).

The WVI can be a dedicated ECU with a suitable wireless interface, can be part of the CGW, or can be realized as plug-in solution, where it is temporarily connected to the vehicle using a suitable interface such as the standardized On-Board Diagnostics (OBD) interface. The plug-in version is of special importance in service centers to ensure backward compatibility for older vehicles without an integrated WVI. As smart gateway, the WVI is not only forwarding data packets, but is rather able to establish a secured connection to the DT, to temporary store a software binary, to support advanced update mechanisms (see Section 5.3), and to allow comprehensive vehicle diagnostics.

4.5.3 Example of a LSU

In the following, an example of an locally-performed software update is used to describe the basic data flow between all involved entities. The activity diagram shown in Figure 4.9 is also showing this procedure.

The DT and the handhelds are typically used within the same environment, such as a service center, for their entire lifetime. Thus, these devices can be considered as already securely connected (i.e., a dedicated pairing process was initially performed) when a vehicle is entering a scenario (e.g., a new vehicle is entering the service center). If this vehicle is equipped with a fully integrated WVI, the vehicle may directly try to establish a secure connection. Otherwise, a user (e.g., a mechanic in a service center) will first connect a plug-in WVI to the vehicle. Next, this WVI will securely connect itself with EASE-UP.

After establishing the connection with the WVI and with the vehicle itself, the DT will request basic information about and from the vehicle. This information is required to retrieve more detailed information of the vehicle from the DT database and, based on this information, also to figure out whether or not new software is available for the vehicle.

Once all entities are connected to each other and the DT found an available software version, the actual update process can be started. A software update will typically be triggered by the user using the handheld device (e.g., in the service center), but in certain cases maybe also by the DT itself. The latter could especially be of importance in highly automated environments such as the assembly line: once a vehicle requiring a new update is entering the transmission range of the software update framework, the DT will start the update process. In the developed architecture, the vehicle itself is not able to trigger the software update itself.

To perform the actual update, first the concerned ECU needs to be prepared to receive new software. To do so, the DT sends suitable requests to the WVI, which is then forwarding these request to the ECU (and also forwards the corresponding ECU responses to the DT). The initialization process thereby also includes an authorization step required to set the ECU to a state where it can actually receive new software. Once the ECU is successfully initiated, the software is transferred to the WVI and in further consequence installed on the ECU. Therefore, EASE-UP supports different approaches and data flows, respectively. More information on these approaches can be found in Section 5.2.1.

The core nodes – DT, handheld, and WVI – must be interconnected in a dependable manner to ensure that a software update can be performed in a secure, fast, as well as reliable way. Therefore, different wireless protocols were compared to each other to find the most suitable wireless network for EASE-UP. More details on the actual choice of the wireless network can be found in Section 5.1.

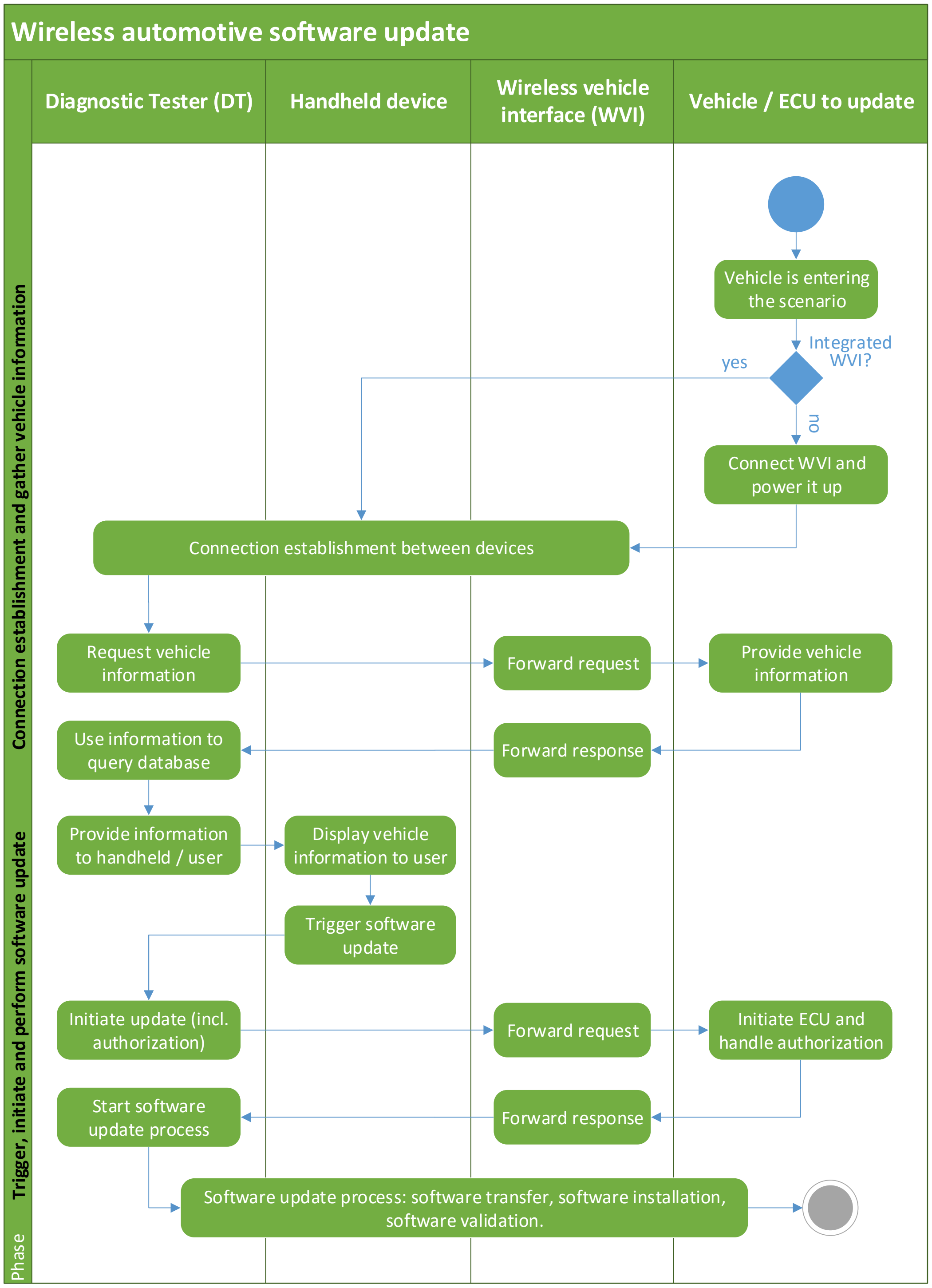

Wireless automotive software update
Diagnostic Tester (DT)
Handheld device
Wireless vehicle interface (WVI)
Vehicle / ECU to update
Phase
Connection establishment and gather vehicle information
Trigger, initiate and perform software update
Vehicle is entering the scenario
Integrated WVI?
yes
no
Connect WVI and power it up
Connection establishment between devices
Request vehicle information
Forward request
Provide vehicle information
Use information to query database
Forward response
Provide information to handheld / user
Display vehicle information to user
Trigger software update
Initiate update (incl. authorization)
Forward request
Initiate ECU and handle authorization
Start software update process
Forward response
Software update process: software transfer, software installation, software validation.

Figure 4.9: Activity diagram showing the basic steps of a generic wireless automotive software update performed in a local environment.

Chapter 5

EASE-UP Networking Aspects

This chapter describes EASE-UP, the developed software update system, in more detail. First different wireless protocols are compared to find the most suitable candidate to be employed as dependable wireless network interconnecting all core entities within the wireless update system, and second, the developed wireless software update protocol as well as its advanced software update features are described.

5.1 Selecting the Wireless Network for LSUs

In this section the most suitable wireless networking technology for the developed wireless software update framework is identified. Therefore, first relevant requirements regarding the wireless networking technology are derived from the system-level requirements which have been stated in Section 4.2. Thereafter, these requirements are used to compare different candidate technologies with each other to find the most suitable wireless networking technology for EASE-UP.

5.1.1 Selection Criteria

The wireless network employed to interconnect the involved users as well as devices in all considered LSU scenarios (please refer to Section 4.1) in a secure, efficient and reliable way, must fulfill certain criteria. These criteria are derived from the system-level requirements listed in Section 4.2 and described in the following [60].

- **Security.** The protection of the wireless network interconnecting all involved entities can be considered as the first layer of security (for more information on the defined security concept see Section 6.2). A suitable wireless networking technology must provide state-of-the-art security mechanisms providing a robust authentication scheme between the involved nodes and must be able to protect the exchanged data by utilizing strong methods protecting data confidentiality and integrity.

- **Efficiency.** Wireless software updates shall be performed as fast as possible independent of the size of the ECU software to transfer and install (typically in a range of some KB up to dozens of MB). Therefore, the wireless protocol shall offer enough *bandwidth*, but must be at least as fast as the automotive CAN bus: 1 MBit/s.

Most software update scenarios are quite dynamic (i.e., vehicles are frequently entering and leaving the system) and the actual updates may take place at different locations (e.g., in the vehicle development phase test vehicles are parked anywhere on the premises). The employed wireless protocol must therefore provide enough *flexibility* to satisfy the peculiarities of different and diverse dynamic LSU scenarios.

Additionally, the *scalability* aspect is very import for several LSU scenarios: in large service centers or assembly lines lots of wireless devices may be interconnected by the wireless network and hence the wireless protocol shall be able to deal with such large scale scenarios encompassing up to 100 nodes within a large building or area.

In the following evaluation of different wireless protocols the efficiency aspect will mainly be discussed w.r.t. *bandwidth*, *flexibility* and *scalability*.

- **Reliability and range**. The wireless network must be able to cover large areas (e.g., large service centers or vehicle assembly lines) and shall be robust against any kind of interference. As updates are mostly performed in harsh environments (for wireless networks), the employed wireless protocol shall offer redundancy to make sure that there is a reliable connection between all involved devices.

- **Adoption in consumer devices**. An additional requirement for EASE-UP is the adoption of the wireless network in consumer devices as the costs for installing a wireless software update system such as EASE-UP shall be kept to a minimum. Therefore, Commercial Off-The-Shelf (COTS) hardware shall be used for realizing the DT as well as the handhelds. The utilized wireless protocol shall hence be supported by most COTS devices.

In Section 5.1 the defined selection criteria are used to choose the most suitable wireless protocol for wireless automotive LSUs.

5.1.2 Potential Wireless Protocols for LSUs

Several wireless protocols can be used to set up a wireless network interconnecting all involved entities of EASE-UP. To find the best fitting protocol, first promising candidates are listed and studied in detail:

- ZigBee (based on IEEE 802.15.4) [30]. ZigBee is a low-power communication protocol based on IEEE 802.15.4 that natively supports different network topologies (i.e., star, tree and mesh). It is heavily used as communication protocol within Wireless Sensor Networks (WSN) due to its low power consumption.

- Bluetooth Low Energy (BLE) [56]. BLE is the successor of classic Bluetooth (not backward-compatible) offering significantly reduced latency and power consumption. While classic Bluetooth is not really scalable due to the utilization of piconets (where only seven slaves can be active at the same time), BLE allows to form networks of unlimited size [66]. BLE is supported by numerous consumer devices such as tablets, smartphones as well as laptops, and is hence used in a wide range of different applications.

- IEEE 802.11n [2]. Wi-Fi networks based on IEEE 802.11n are available in most company buildings, universities and private homes. Wi-Fi offers high bandwidths (i.e., up to 600 Mbps for IEEE 802.11n), and is supported by most smartphones, tablets and laptops. Wi-Fi utilizes a star topology coordinated by an Access Point (AP) to interconnect all devices within a network. An example of a Wi-Fi network topology is shown in Fig. 5.1a, where a DT acting as AP interconnects entities within a local software update scenario.

- IEEE 802.11p. the communication between vehicles amongst themselves (i.e., vehicle-to-vehicle communication) as well as between vehicles and roadside infrastructure (i.e., vehicle-to-infrastructure communication) such as traffic lights is specified in several automotive standards. The underlying wireless communication is standardized in IEEE 802.11p. Although this standard already exists since several years (first version published in 2010), it is not integrated in today's vehicles yet, and currently substantial research is performed in the field of 5G with the goal of replacing IEEE 802.11p. Therefore, it seems unclear if IEEE 802.11p will be used in future vehicles at all and hence it is not further discussed in the following comparison.

- IEEE 802.11s [29]: contrary to classic Wi-Fi networks (e.g., IEEE 802.11n), IEEE 802.11s uses a mesh topology to interconnect all entities within a network while supporting rates similar to IEEE 802.11n (same hardware is used; routing is done at the MAC layer). An example for such an IEEE 802.11s-based mesh network is shown in Fig. 5.1b: nodes within the transmission range establish a (secured) link with each other and, on top, maintain a path map, where the best route to all nodes within the mesh network is stored. If no reliable direct connection between two nodes within a network can be established (e.g., the nodes are too far apart from each other), other nodes in between will forward the exchanged packets.

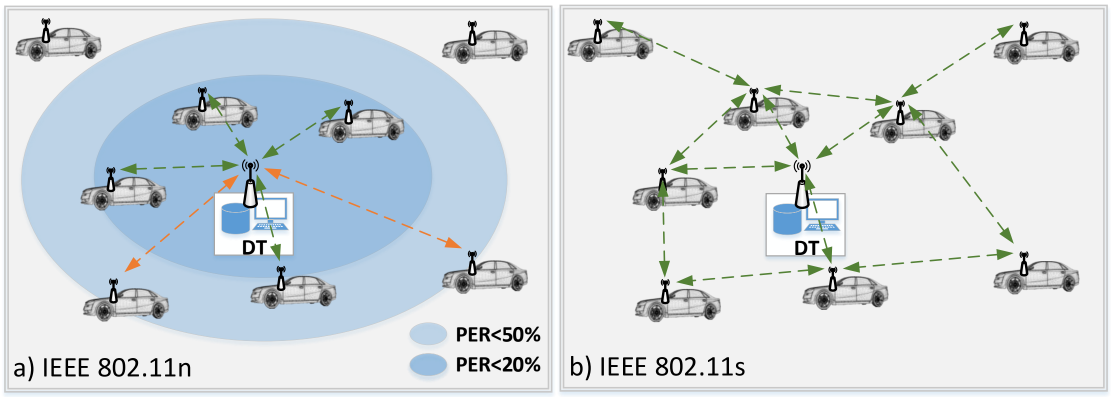


Figure 5.1: IEEE 802.11n (a) and IEEE 802.11s (b) are employed in the same example scenario. Using IEEE 802.11n leads to isolated vehicles (i.e., vehicles without a reliable connection to the DT) while IEEE 802.11s is able to interconnect all involved entities.

5.1.3 Comparison of Wireless Protocols

Next, these aforementioned protocols are compared with each other w.r.t. selection criteria defined in Section 5.1.1.

Security. IEEE 802.11n as well as IEEE 802.11s offer advanced security measures and configurable options providing suitable security solutions for private home applications as well as for large enterprises. Although critical security vulnerabilities (e.g., eavesdropping and man-in-the-middle attacks were possible in BLE version 4.1 and below) of BLE were fixed in version 4.2 and above [23], BLE is still not considered to be as secure as Wi-Fi. The ZigBee standard itself implements quite strong security features, but still has some vulnerabilities (e.g., key initialization and distribution) [17, 65], even in the latest version, ZigBee 3.0 [40]. Thus, the security aspect is not yet satisfied in current ZigBee implementations.

Bandwidth. ZigBee and BLE are particularly advantageous in low-power applications and for battery-driven devices, as the employed communication mechanisms are very power-efficient (e.g., compared to Wi-Fi). However, power-efficiency often, as in case of ZigBee and BLE, comes at the cost of low data rates: while IEEE 802.11n and IEEE 802.11s are providing high bandwidth (per channel) with up to 300 Mbps, BLE and ZigBee are only offering 1 Mbps and 250 Kbps [38], respectively.

Reliability and range. Another important aspect is the reliability of the utilized wireless network and especially its range (i.e., the ability of reaching all nodes of a network with at least one reliable link). Therefore, the range of a wireless node, but even more importantly of the wireless network in its entirely, is evaluated w.r.t the discussed wireless candidates: ZigBee and BLE nodes are mostly designed and implemented to be as power-efficient as possible. Thus, the used HW chipsets are often only supporting low transmission output power (e.g., 0 dBm) while Wi-Fi hardware can provide TX output level up to 20 dBm (and even more). Therefore, a Wi-Fi node typically offers a significantly wider transmission range compared to BLE and ZigBee devices. However, as mentioned above, the more important aspect is the range of the entire network. IEEE 802.11n and BLE are using star topologies to interconnect devices and the AP (i.e., in IEEE 802.11n) or the master node (i.e., in BLE). If a device/slave is too far away, there will be no (reliable) connection with the network. In mesh networks (ZigBee and IEEE 802.11s), in contrast, each node with at least one other node of the network in its transmission range, can be added to the wireless mesh network. A packet intended to be sent from one edge of a mesh network to the other will thereby be forwarded by nodes in the middle of the network (i.e., multi-hop communication). Please note that BLE 5.0 and above will also support mesh networks. However, at the time of writing BLE mesh networks are not yet supported by most chipsets.

A summary of this comparison can be found in Table 5.1, showing that IEEE 802.11s is the most promising candidate, as it fits best the defined selection criteria. The *adoption in COTS* aspect is the only aspect that is not fully satisfied by IEEE 802.11s, as its software implementation (i.e., mainly to provide routing on MAC layer) is not available for all chipsets yet. Further information on the choice of IEEE 802.11s as well as some experiments showing its applicability for automotive applications can be found in [60].

	ZigBee	BLE (v4.2)	IEEE 802.11n	IEEE 802.11p	IEEE 802.11s
Security	●	●●	●●●	●●	●●●
Bandwidth	●	●●	●●●	●●	●●●
Scalability	●●●	●●	●●	●●	●●●
Flexibility	●●●	●●	●	●●●	●●●
Reliability	●●	●	●●	●●	●●●
Ad. in COTS	●	●●●	●●●	●	●●

Table 5.1: Comparison of wireless protocols w.r.t. the defined selection aspects. Per aspect and per wireless protocol one to three dots are indicating whether the protocol can fulfill the aspect insufficiently, sufficiently, or fully sufficiently, respectively.

5.1.4 IEEE 802.11s as Selected Wireless Network for LSUs

Based on the comparison of different wireless technologies described in the previous sections, IEEE 802.11s was identified as the most suitable wireless network for automotive LSUs. In this section, the functionality of IEEE 802.11s is briefly described and its core features are listed.

As sketched in Figure 5.2, IEEE 802.11s networks utilize a mesh topology to interconnect all involved nodes. In its basic mode, each IEEE 802.11s node is equal w.r.t. the network. However, IEEE 802.11s also allows to assign certain nodes as *root nodes*. Such a root node will proactively announce itself within the network, allowing other nodes to maintain the best (multi-hop) path to it. A root node is often also working as *gate node*. The latter interconnects the mesh network and other networks (e.g., a LAN) by forwarding data sent from the mesh network to entities outside the network (i.e., if a mesh network cannot resolve an address within the mesh network, it will send the packet to the gate node) and vice versa.

Routing in IEEE 802.11s. Within an IEEE 802.11s network one has to distinguish between *links* and *paths*. Links are established locally between two nodes in transmission range with each other (i.e., solid blue lines in Figure 5.2). Therefore, each node periodically broadcasts its address and the ID of the network it is in. If a new node receives such a broadcast beacon, it uses the contained information to establish a link to the node sending the beacon (i.e., three-way-handshake). Theoretically, a node can have an infinite number of links to neighboring nodes. However, the size of the neighbor-table holding the established links is limited in practice (e.g., a maximum of 8 neighbors is allowed for the TP-Link TL-WN722N modules used for the developed prototypes). In the worst case this limitation can lead to isolated nodes (see also Section 7.2.2).

Links can be used for direct communication between two nodes. However, one core feature of IEEE 802.11s is its multi-hop capability. The latter allows two nodes which are not in direct communication range with each other (e.g., nodes 4 and 9 in Fig. 5.2), to still exchange messages in a dependable way if a so-called *path* can be found between these two nodes. Paths are maintained on top of the established links and the IEEE 802.11s standard allows different ways on how to perform the routing (e.g., the identification of new paths as well as the maintenance of established ones). The most widely used algorithm in IEEE 802.11s networks is the Hybrid Wireless Mesh Protocol (HWMP). This protocol provides mechanisms to find the best (i.e., most reliable and smallest latency) path through the network. Therefore, a node maintains a second table where, for each target node, the next hop (i.e., the first node within the path) is stored. Data packets are always sent via paths and hence packets are sometimes transferred via a multi-hop path even if a direct link between two nodes is established (especially for weak links). For example, in Figure 5.2, packets sent from node 3 to node 7 are transferred via node 5.

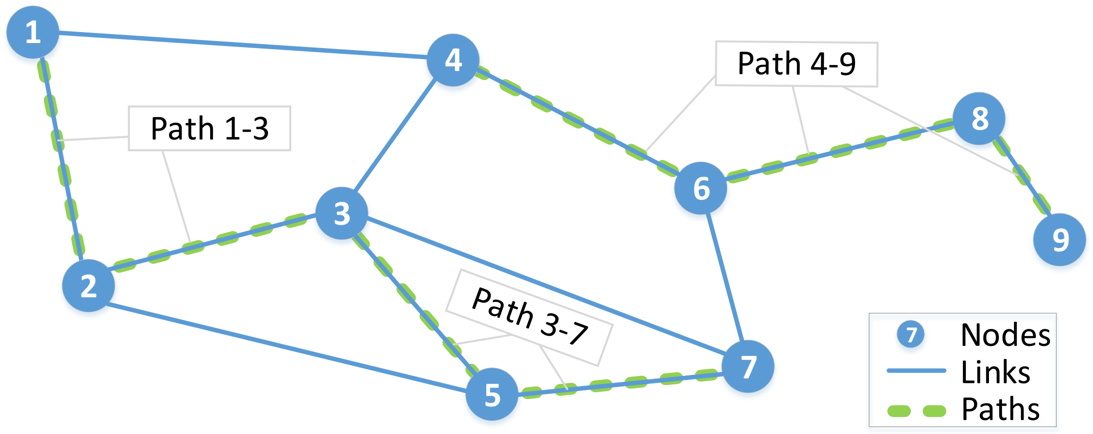


Figure 5.2: Example of an IEEE 802.11s mesh network encompassing 9 nodes.

Security in IEEE 802.11s. The IEEE 802.11s standard provides a dedicated security protocol – Simultaneous Authentication of Equals (SAE) [25] – handling the secure key exchange in mesh networks. SAE hence provides security measures to protect the mesh network itself, as well as the transferred data using a pre-shared password to handle the secure key exchange. SAE is supported by the most popular open-source IEEE 802.11s implementation open11s [11] and also by wpa_supplicant, a generic security framework for different types of wireless networks (e.g., popular in different Linux distributions).

SAE is protecting the network layer of the developed wireless automotive software update framework and thus is an important aspect within the defined security concept (i.e., first layer of security). More information on the defined comprehensive security concept can be found in Section 6.2 and further details on the impact of SAE on network layer performance is given in Section 7.2.3.

5.2 Wireless Software Update Protocol

In this section the actual protocol allowing wireless automotive LSUs is described. This protocol is used by EASE-UP to support all steps from establishing a trustable connection between the core entities (i.e., WVI, DT, and the handheld devices) to finally installing the new software binary on the ECU. The protocol is employing several automotive standards to support a wide range of ECUs (i.e., ECUs from different suppliers as well as different types of ECUs) and to ensure that the developed solution is compatible with vehicles from different OEMs as well as vehicles in a different age (i.e., ensuring backward compatibility).

Several automotive standards are supporting the software update process or are describing the underlying communication protocols. In Section 2.2 the most important standards are discussed and an overview on the diagnostic stack is given.

5.2.1 Basic Software Update Protocol

EASE-UP's basic wireless software update protocol employs the different automotive protocols (see Section 2.2 for details) to handle six major steps as sketched in Figure 5.3. In the following, these steps are listed and described in more detail. Please note that the following descriptions are focusing on the data flow between the involved entities. Security aspects are addressed in Section 6.2 where EASE-UP's security concept is presented.

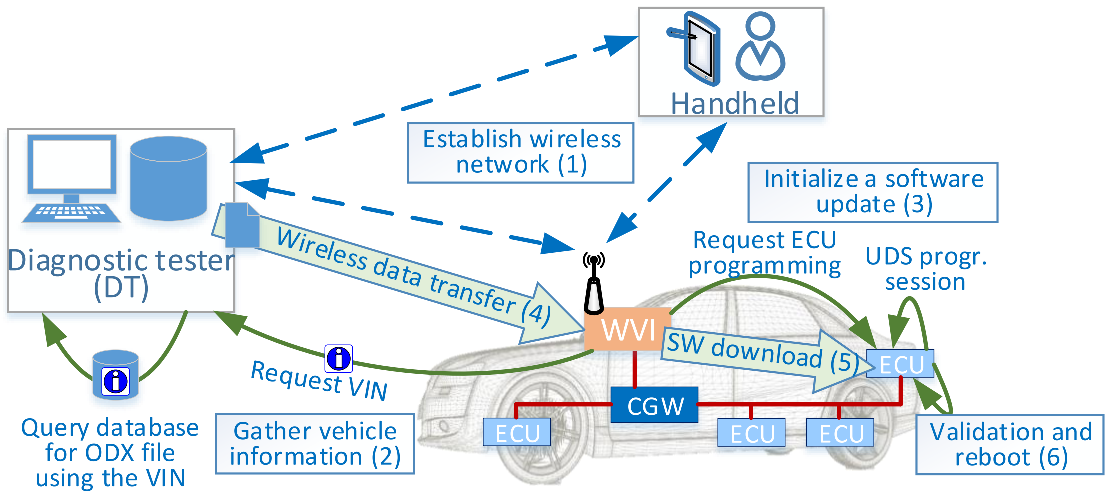

Handheld
Establish wireless network (1)
Diagnostic tester (DT)
Initialize a software update (3)
Request ECU programming
UDS progr. session
Wireless data transfer (4)
WVI
SW download (5)
ECU
CGW
ECU
ECU
ECU
Request VIN
Query database for ODX file using the VIN
Gather vehicle information (2)
Validation and reboot (6)

Figure 5.3: The defined wireless update protocol. Six dedicated steps are employed to control the entire wireless software update process for automotive ECUs.

Step 1 – Discovery and connection establishment. This step refers to the establishment of a wireless network interconnecting DTs, WVIs, and handhelds. Therefore, the DT periodically broadcasts beacons containing information about the available diagnostic network as well as the provided services (e.g., a software update for specific brands). A WVI will use such a beacon to discover the available diagnostic network and connect to it if relevant services are offered (i.e., the DT supports the brand of the vehicle). Strong authentication mechanisms are employed in the connection establishment process (see 6.2 for more information).

Step 2 – Gather vehicle information. In this step, the DT requests basic vehicle information from the WVI and use it to query the DT database. The Vehicle Identification Number (VIN) requested by the DT from the vehicle (in particular its WVI) will be used to query the DT's local ODX database or, if no information is available locally, to request detailed vehicle descriptions via the OEM backbone network. The DT will parse the ODX file containing the vehicle descriptions and use the developed software update protocol to distribute relevant information about the vehicle to the connected handhelds.

Step 3 – Initialize a software update. In this step, the software update is initialized on ECU level, if new software is available and shall be installed (e.g., update is triggered by the user). The user (e.g., a mechanic in a service center) will be informed about an available software update for a vehicle and decides when the update shall be installed. Once the update is started, the developed protocol is used to inform all involved entities about the update and to distribute relevant information such as the physical and logical bus ID of the ECU required by the WVI to initialize the software update on ECU level using UDS. The authorization of the software update is handled by the DT. Hence, neither the WVI nor the handhelds obtain information about the authorization key due to security reasons (see Section 6.2 for more information).

Step 4 – Wireless data transfer. This step describes the transfer of the software binary from the DT to the WVI using the wireless network. The developed protocol provides an efficient, reliable and secure data transfer of the software binary from the DT to the WVI using the wireless network. Therefore, the software binary to transfer is divided into data chunks of a configurable size. Next, dedicated security mechanisms ensuring data integrity as well as confidentiality are employed and the resulting encrypted data chunk is placed in the payload section of a newly created data packet. Finally, the data packet is sent from the DT to the WVI using TCP or UDP (configurable). To detect out-of-order as well as missing packets, each packet is equipped with an unique identifier and a packet number. The WVI will send an acknowledgment packet back to the DT for each received data packet.

Step 5 – Software download. This step describes the download of the software binary from the WVI to the ECU using UDS (typical on CAN). The wireless software update protocol provides mechanisms to trigger and monitor the UDS-based data download from the WVI to the ECU. The user can employ these mechanisms to request the current state (e.g., number of bytes transferred or time remaining) of the download process.

A software binary typically consists of several code segments. Each code segment has a dedicated start address w.r.t. the location within the flash memory of an ECU. UDS allows to specify this start address along with the size of the current code segment within a *Request Download* frame sent from the WVI to the ECU. Once this frame was acknowledged by the ECU, the WVI starts to download the actual code bytes of the segment using the UDS *Transfer Data* frames. The ECU will directly write the received bytes into its flash memory starting from the specified start address until the specified size of the code segment is reached.

Step 6 – Validation and reboot. In the last step, the software update is validated using a dedicated UDS frame such as *Routine Control* and the ECU is rebooted.

Protocol based on a state machine. The protocol is utilizing a state machine scheme representing all six steps. Each step is represented by a dedicated state and certain substates are defined for each state as shown in Figure 5.4. The state machine pattern is chosen to prevent system failures due to actions performed in a wrong stage (e.g., starting a data download while the ECU is not yet initialized) and is also used within the developed command structure. The latter is employed by the DT to control the software update process. Each defined command is dedicated to a specific state and has an unique identifier that is present in every command message exchanged between the DT and the WVI. Besides the command identifier, a command message also optionally contains metadata such as the seed bytes generated by the ECU.

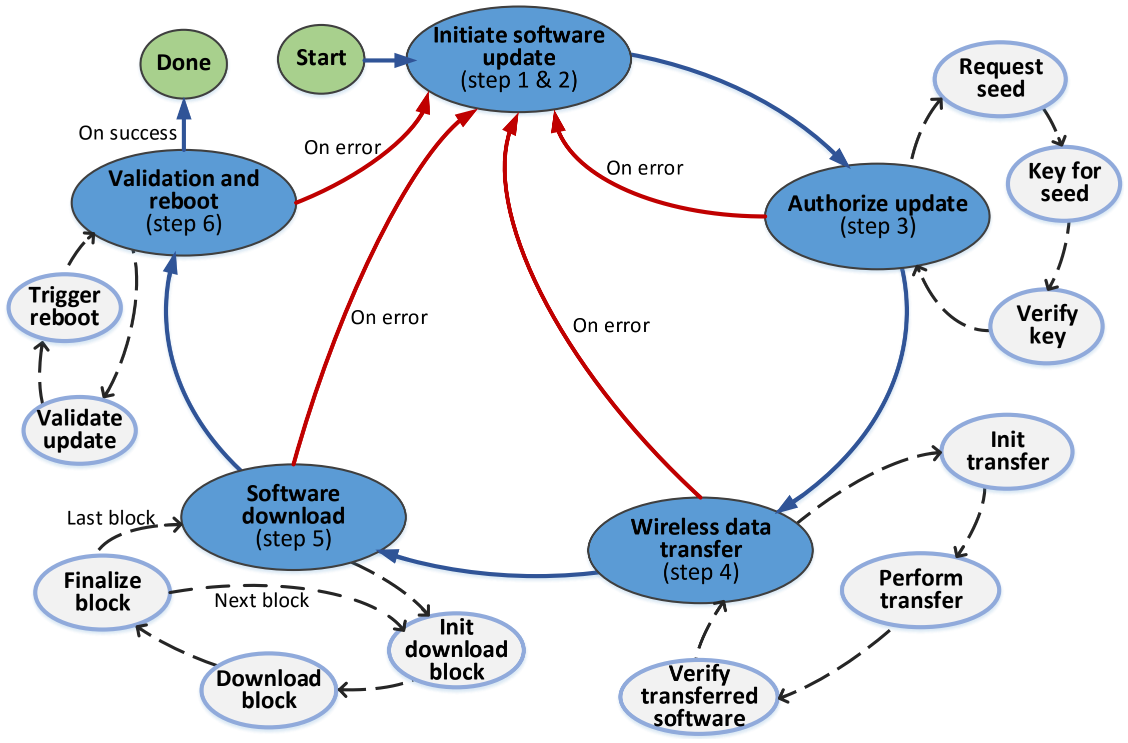


Figure 5.4: State machine used on the WVI. Each step of the wireless software update protocol is represented (please note that step 1 and 2 are handled together within the first state) and also the substates are shown. A similar state machine is used on the DT.

The defined states are also used for profiling the timing behavior of the wireless software update process. Further information on these measurements and the gathered results can be found in Section 7.2.1.

Supported update modes. EASE-UP's basic wireless software update protocol supports different software update modes. These modes are mainly relevant for the steps 4 (*Wireless data transfer*) and 5 (*Software download*), and describe how software transfer from the DT to the ECU via the WVI is handled.

- **Direct programming.** In this mode, the software binary is parsed on the DT and transferred to CAN frames. These CAN frames are then wrapped into a predefined packet, then sent to the WVI, where the CAN frame is unwrapped and finally sent to the ECU using the CAN bus. For each frame, the WVI waits for the acknowledgment frame from the ECU and forwards it to the DT. Then the next CAN frame is created by the DT sent to the WVI and forwarded to the ECU (the WVI is acting as wireless-to-wired gateway in this mode). This mode is mainly required if an ECU is not supporting standard-conform UDS-programming.

- **Two-stage programming.** In this mode, the software binary is first transferred to the WVI and in a second step downloaded to the ECU. Therefore, this mode allows to transfer significantly larger data frames and is more efficient and faster w.r.t. the wireless data transfer. Furthermore, this mode allows the WVI to work more autonomously and thereby reduces the load of the wireless network, as significantly less packets must be exchanged.

By default, EASE-UP utilizes the *two-stage programming* mode due to its significant advantages compared to the *direct programming* mode.

5.3 Advanced Software Update Mechanisms

In the previous section EASE-UP's basic wireless software update protocol was described in detail. This protocol already significantly increases the flexibility of the developed system compared to wired solutions (e.g., a mechanic can work on several vehicles without disconnecting from the current vehicle and connecting to another one). However, there is no improvement w.r.t. the overall software update duration compared to a wired solution. In fact, the defined wireless software update protocol leads to an additional protocol overhead of about 12% (see Section 7.2.1 for more details) compared to a purely wired approach, as the wired software download via CAN must be wrapped within the wireless solution. To increase the efficiency of the wireless solution (i.e., reducing the duration of the update process), advanced software update mechanisms such as parallel updates (Section 5.3.1) and partial updates (Section 5.3.2) have been developed. Results showing the benefits of employing these advanced mechanisms can be found in Section 7.2.1.

5.3.1 Parallel Software Update

Parallel software updates are advantageous in all scenarios where several ECUs integrated in different vehicles require the same new software binary: the assembly line (where dozens of new vehicles of the same type are produced within a short time range), the service center (where multiple vehicles require new software due to an identified software bug), or in the vehicle development (where a fleet of test vehicles receives the same newly developed software for a prototype ECU).

EASE-UP supports parallel software updates applicable for all aforementioned LSU scenarios, as it allows to perform several steps of the defined update protocol in parallel. In particular, the update protocol provides a specific mechanism to register several vehicles for a LSU. As shown in Figure 5.5, EASE-UP provide mechanisms to perform a parallel update (especially w.r.t. steps 3 to 6) for several vehicles at the same time: two vehicles are connected to a DT. The gathered information shows that a parallel software update is possible. Once the vehicles are ready to receive the new software (e.g., as the manual tasks are done), the parallel software update process can be started (e.g., triggered by the user) and most of the previously described steps (see Section 5.2.1) are performed in parallel. Please note that not all steps can be done fully in parallel (especially security-related steps; see Section 6.2) as they require exclusive access to specific software or hardware components (e.g., a certain hardware security module on the DT). The system is developed in a way that all involved ECUs/WVIs must complete the current step (e.g., initialize a software update – step 3) before the next step (e.g., wireless data transfer – step 4) is triggered by the DT. In the following, all relevant steps (i.e., starting from step

3) are briefly discussed to describe which extensions have been made to allow parallel updates. Additionally, the parallel update process for two vehicles is illustrated in Figures 5.5 and 5.6.

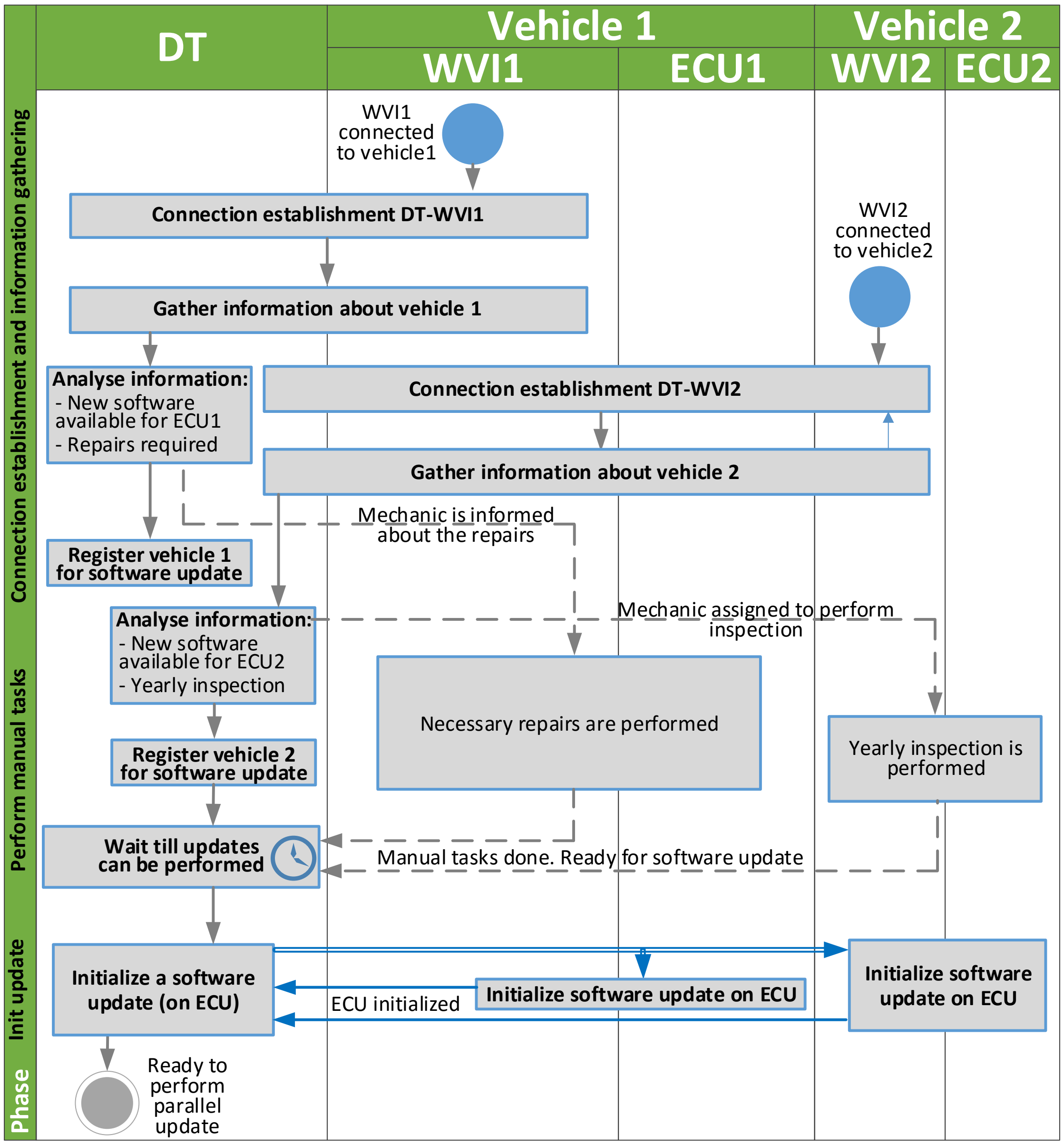

DT
Vehicle 1
WVI1
ECU1
Vehicle 2
WVI2
ECU2
Connection establishment and information gathering
Perform manual tasks
Init update
Phase
WVI1 connected to vehicle1
Connection establishment DT-WVI1
Gather information about vehicle 1
WVI2 connected to vehicle2
Analyse information: - New software available for ECU1 - Repairs required
Connection establishment DT-WVI2
Gather information about vehicle 2
Mechanic is informed about the repairs
Register vehicle 1 for software update
Analyse information: - New software available for ECU2 - Yearly inspection
Mechanic assigned to perform inspection
Necessary repairs are performed
Yearly inspection is performed
Register vehicle 2 for software update
Wait till updates can be performed
Manual tasks done. Ready for software update
Initialize a software update (on ECU)
ECU initialized
Initialize software update on ECU
Initialize software update on ECU
Ready to perform parallel update

Figure 5.5: Parallel software update process. Information about the vehicles is gathered to identify possibilities to perform an update for several vehicles in parallel.

Once all involved vehicles are securely connected to EASE-UP (e.g., within an assembly line) and the required information about the vehicle is gathered, the parallel wireless software update can be started (see Figure 5.5). Therefore, the DT will trigger the *initialize a software update* step on the concerned ECUs by sending a multicast message (i.e., a command message) to the involved WVIs. These WVIs individually take care about

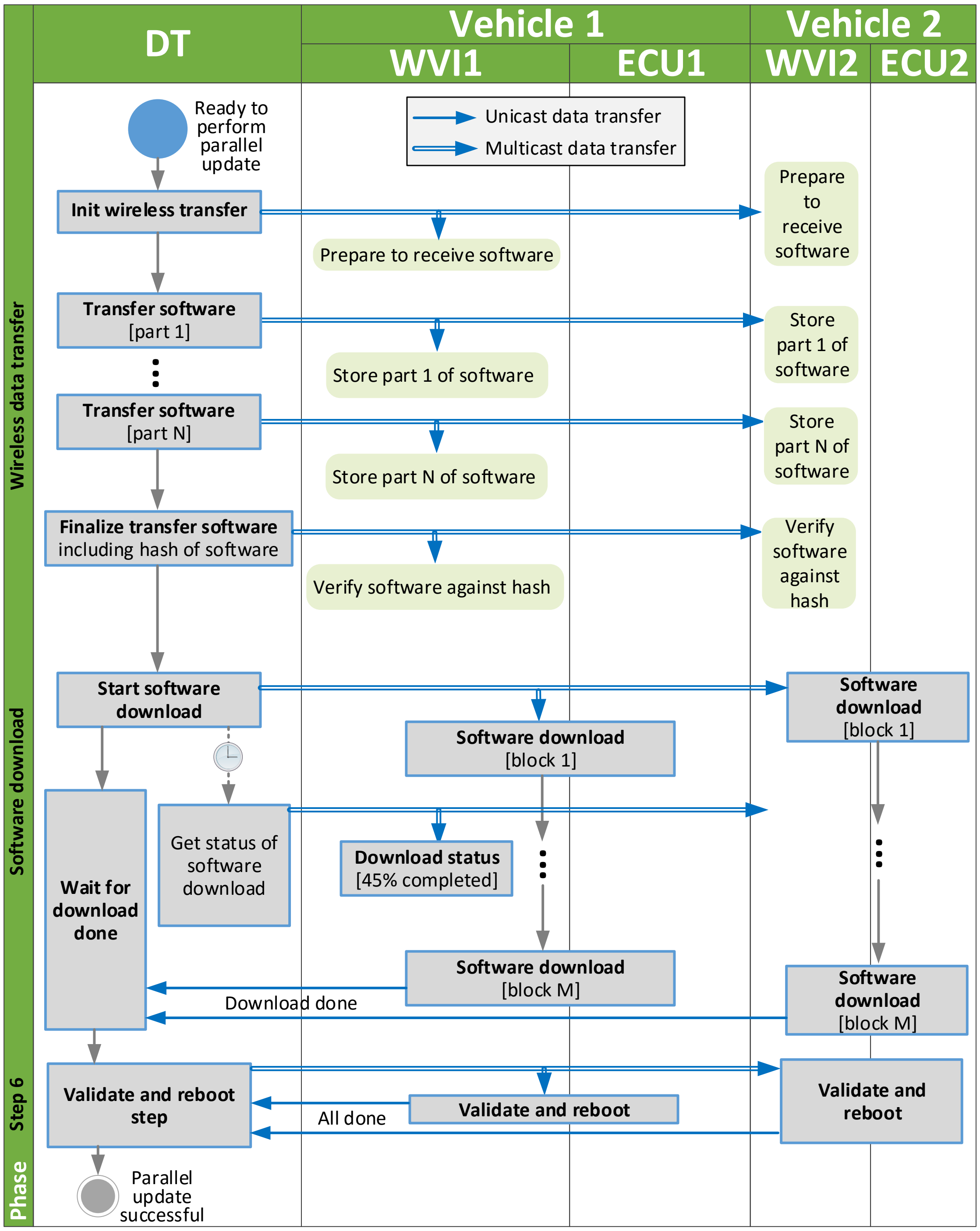

DT
Vehicle 1
WVI1
ECU1
Vehicle 2
WVI2
ECU2
Phase
Wireless data transfer
Software download
Step 6
Ready to perform parallel update
Unicast data transfer
Multicast data transfer
Init wireless transfer
Prepare to receive software
Prepare to receive software
Transfer software [part 1]
Store part 1 of software
Store part 1 of software
Transfer software [part N]
Store part N of software
Store part N of software
Finalize transfer software including hash of software
Verify software against hash
Verify software against hash
Start software download
Software download [block 1]
Software download [block 1]
Wait for download done
Get status of software download
Download status [45% completed]
Software download [block M]
Software download [block M]
Download done
Validate and reboot step
Validate and reboot
Validate and reboot
All done
Parallel update successful

Figure 5.6: Parallel software update process. The software is first transferred from the DT to the WVIs and then installed on the ECUs. The software download step is handled by each WVI individually. The DT can poll the current status of the download and will also be informed once the download is done.

the update initialization on ECU level and directly communicate with the DT (i.e., unicast data stream), if required (e.g., for handling the Seed&Key mechanisms). Once the initialization procedure is done, the WVIs acknowledge the received multicast command message and wait for further commands coming from the DT.

The DT waits until each WVI has successfully handled the update initialization process and then triggers the *wireless data transfer* of the software binary. As shown in Figure 5.6, a multicast is used to send the binary from the DT to each WVI simultaneously. Therefore, the DT uses a predefined multicast address instead of the address of a specific WVI. No further adaptions are required in this step.

Once all WVIs have informed the DT that they have received the new software binary, the DT triggers the *software download* by sending another multicast command message to all involved WVIs. Next, these WVIs individually handle the software download process to the ECU (e.g., over CAN) and inform the DT once the software download is done. The DT can asynchronously request the current status of the download process by sending a dedicated command message to one or all involved ECUs.

In the last step, the DT uses another multicast command to trigger the ECU validation and finally the reboot of the ECU. Again, the only difference to the basic software update approach is that a multicast is used instead of an unicast.

EASE-UP's parallel software update mode is rather simple to develop, as the basic update protocol was already designed in a way that it allows certain steps of the update process to be performed simultaneously. However, from security point of view, this mode requires some more attention, as certain features such as the secure multicast (e.g., transfer the binary to several WVIs simultaneously) demand extensions to the basic security concept. More information on these features is given in Section 6.2.

5.3.2 Partial Software Update

Partial software updates are particularly advantageous if the current and the previous software version of an ECU only differ slightly (e.g., only one parameter field is updated) and especially if the software binary itself is rather large (e.g., some megabytes). In such a case, a partial software update will allow to significantly reduce the number of bytes transferred, as only a small delta (i.e., at a minimum the bytes which are different from one version to the other) must be sent from the DT to the ECU via the WVI. Hence, the overall duration of the software update process can be reduced.

Partial software updates are already used in telecommunication (i.e., delta updates for smartphones) as well as Wireless Sensor Networks (WSN). In Section 3.3 relevant approaches are discussed.

Partial software updates for automotive ECUs. In contrast to parallel software updates, which are completely independent from the ECU to be updated, the efficacy of partial software updates strongly depend on the underlying ECU. First, the ECU must be able to support partial software updates at all, and second, the utilized flash architecture as well as the used memory layout of the developed ECU software significantly impact the efficiency of the partial software update process. Hence, in the following the developed partial software update process for one specific automotive ECU – the Infineon AURIX ECU – will be described in more detail (see also [37]). However, the described concepts can of course also be used for other ECU platforms (in slightly adapted form).

Developed partial software update mechanism. The developed mechanism exploits the ability of UDS to update single code segments individually. If only one code segment of an ECU software binary is affected by a software update (e.g., due to a bug fix within this code segment), it is enough to only update this code segment instead of the entire software binary. This basic idea requires the DT to be able to detect situations where only some code segments are affected by a software update. In the developed software update system, the software binary parser of the DT also allows to compare two software versions with each other, resulting in a *partial file* only containing the affected code segments.

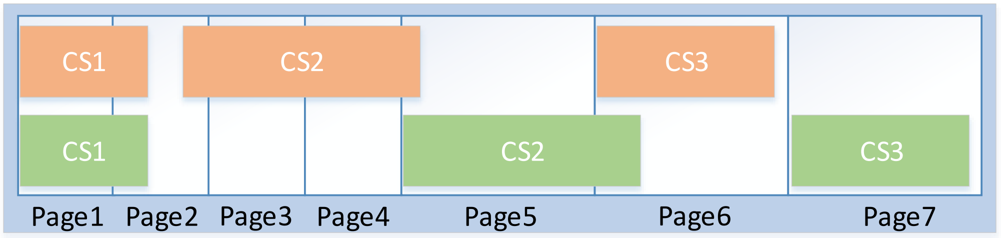


Figure 5.7: Software binary consisting of three Code Segments (CSs) in the ECU memory. The upper view shows a bad placement of the CSs w.r.t. partial software updates. The lower placement of the CSs is more suitable for partial software updates: no CSs are located within the same page and they are placed in the memory according to their size.

The flash memory of an ECU is typically divided into physical blocks (so-called *pages*) and the employed flash controller often only allows to delete and write an entire page. Therefore, the location of the code segment within the physical memory must be taken into account when developing the partial software update mechanism within the bootloader. In Figure 5.7, an example of a software binary consisting of three Code Segments (CSs) and their location within the ECU memory is given. In a case where two CSs are placed within the same page (e.g., CS1 and CS2 in Figure 5.7, upper illustration), both CSs have to be written to the affected page even if only one of the CSs has changed. The given bad placement example would require to rewrite five pages in total if CS1 or CS2 require an update. The developed ECU bootloader, but also the DT and especially its parser, have to take these placement issues into account and ensure that all affected pages are filled with all concerned CSs. Therefore, the developed DT prototype compares the parsed binary (its defined CSs) with the memory layout of the ECU to ensure that the *partial file* does not only contain the changed CSs, but also other CSs that needs to be written to the ECU again due to the placement issue described above.

EASE-UP's wireless software update protocol supports partial software updates without any adaptions, as all the required update features are provided by the DT itself. Other components such as the WVI or certain steps of the software update process (e.g., the wireless data transfer) are not affected at all, and also the defined security concept does not require any adaption. In Figure 5.8, the designed partial update mechanism consisting of five major steps is illustrated. First, vehicle- and ECU-specific information is collected. In a second step, this information is used to check if new software for the ECUs of the vehicle is available. In case new software is found, the so-called *diff tool* of the DT is used to compare the currently installed (i.e., old) software version with the new software (i.e., step 3). As a result of this comparison, the *partial file* is created (i.e., step 4). The created *partial file* includes all CSs, which have to be transferred and installed on the ECU (i.e., changed as well as affected CSs; see description above). Finally, in step 5, the *partial file* is transferred to the ECU and ultimately installed on it. In the best case, the *partial file* only includes the actually changed CS, and in the worst case all CS of the new software have to be transferred and installed (i.e., in worst case, the partial file is equal to the new software binary). It is important to mention that the developed partial update mechanism does not require any adaption to the UDS standard or the ECU. Therefore, the mechanism can basically be used for all ECUs supporting UDS. However, the actual benefit of employing the developed mechanism heavily relies on the memory layout of the ECU and the placement of the CS within the memory.

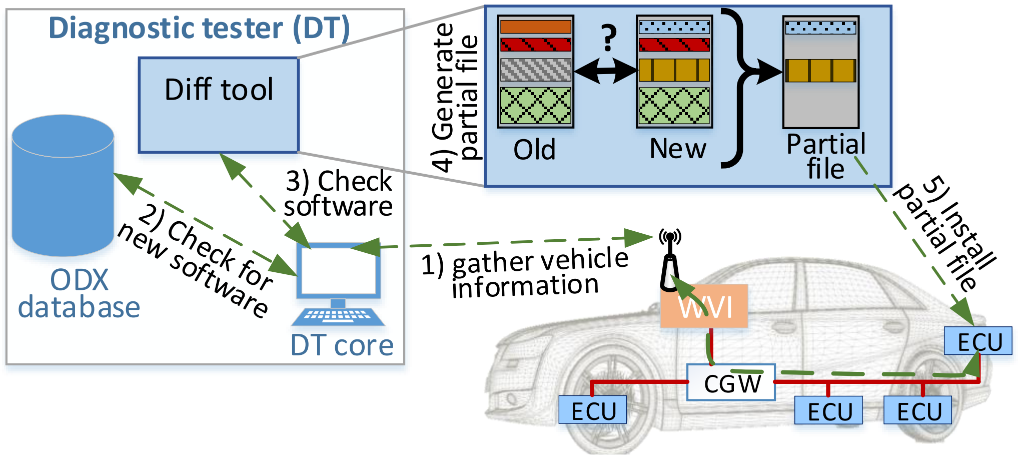


Figure 5.8: Partial software update approach. Five major steps are required to perform a partial software update in EASE-UP.

Guidelines for efficient partial software updates. Partial software updates can significantly reduce the time required to perform software updates and hence increase the efficiency of the entire system. Evaluation results showing such improvements are presented in Section 7.2.1. The update duration is strongly affected by the placement of the code segments of an ECU software binary. Thus, some basic guidelines for the ECU software development and the placement of the resulting code segments are presented next. These guidelines are not dedicated to any specific ECU, but are applicable for a generic ECU platform where the flash memory is divided into physical pages and logical memory sections are defined within the development environment (i.e., logical memory layout). Such a section covers one or several pages of the ECU flash memory.

In Figure 5.9 an example of an ECU flash memory layout is shown. The illustrated layout encompasses several sections of different sizes. Most development environments allow to adapt the default memory layout. This can be very useful especially if a very large (e.g., for a comprehensive code segment) or lots of small sections (e.g., if many small code segments will be implemented) are required. However, one has to make sure that the changed memory layout still corresponds to the underlying physical pages (i.e., a section can cover several pages but shall always end at the end of a page). Another important aspect is the placement of the single code segments as indicated in Figure 5.9: large code segments that will remain the same over the lifetime of a vehicle shall be placed in large sections. Parameters or other code fragments more likely to change over the lifetime of the ECU shall be placed in rather small sections. This placement will significantly improve the performance of partial software updates.

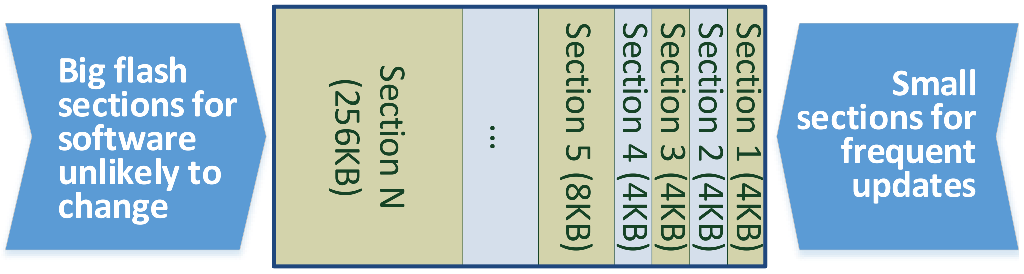


Figure 5.9: The flash memory of an ECU is divided into sections with different block sizes. Figure taken from [37].

Chapter 6

EASE-UP Security Aspects

Security is a crucial aspect of a wireless automotive software update system and for almost all automotive applications in general. The potential security threats range from endangering the privacy of the vehicle owner/driver by stealing personal information or tracking the vehicle, over theft of the vehicle or personal belongings within the vehicle, to endangering the safety of the driver, the passengers as well as other road users by gaining remote control over the vehicle or installing malicious software on an ECU.

To overcome this wide range of attacks, the automotive industry started to define comprehensive security design processes for vehicle applications. These processes are thereby often similar to the well-established automotive safety counterparts described in the ISO 26262 [33] standard. A first security standard – SAE J3061 [52] – is describing a security design process for cyber-physical vehicle systems. However, the guidelines presented by this standard are only specifying the conceptual steps towards a secure automotive application, but do not provide any details on how to tackle these single steps.

In this chapter a measurable security design approach applicable to support certain steps described in SAE J3061 is proposed (Section 6.1) and used to find the best security concept for a wireless software update system applicable for all aforementioned local update scenarios (see Section 4.1). This comprehensive security concept is described in more detail in Section 6.2 and finally evaluated in Section 6.2.6.

6.1 Measurable Security Design Approach

The proposed security design approach – the DEWI[1] Security Metric [62] – is based on the SHIELD Multi-Metrics [21, 45], a measurable security design approach for analyzing complex cyber-physical systems. The DEWI Security Metric can be used to analyze and design a system w.r.t. to security and thereby support the system design flow defined in the SAE security standard SAE J3061 [52].

In Chapter 2 related background information on the SHIELD Multi-Metrics (see Section 2.5) as well as the new SAE security standard for automotive systems (see Section 2.4) is provided. Next, the DEWI Security Metric is described in more detail, and an example on how to apply this metric is provided. Additionally, an evaluation of the DEWI

[1] EU ARTEMIS project Dependable Embedded Wireless Infrastructure (DEWI). For further details see www.dewi-project.eu.

Security Metric is performed.

6.1.1 DEWI Security Metric

The SHIELD Multi-Metrics approach (see Section 2.5) was successfully applied in the DEWI project to analyze and evaluate different application scenarios. However, some limitations of this approach were identified and hence the DEWI Security Metric was defined to overcome these limitations and foster the use of a measurable security approach in real-world applications.

SHIELD's limitations and their solutions by the DEWI Security Metric. The identified limitations range from a subjective assignment of the security goals as well as security values for the system components leading to non-replicable results, over the individual analysis of privacy aspects, to significant difficulties when evaluating a system w.r.t the mixed term dependability. In the following the most important limitations are discussed and the corresponding solution within the DEWI Security Metric is described.

Rating of system parameters: the definition of a metric consisting of a weight and a criticality value for each identified system parameter is a crucial step within the SHIELD Multi-Metrics approach. However, the authors of the latter do not give any guidance on how these values shall be chosen. This can potentially lead to incomparable and non-replicable results as users of the approach (i.e., security experts) will employ different (subjective) rating schemes.

To mitigate this issue, the DEWI Security Metric comes with a suitable scale and corresponding guidelines. This scale supports the security experts to choose the score values for the system parameters and to comprehend the choice of their colleagues (i.e., when evaluating the previously chosen values). A non-linear scale was defined for the parameter assessment as it better fits the human interpretation of criticality (security value = 100 - criticality value). More information can be found in [62, 5].

Choosing the goal values: the goal values within the SHIELD Multi-Metrics approach represent the required security level of an application (e.g., a wireless software update system) in a specific scenario (e.g., updates in an assembly line). The chosen goal values can easily be compared to the results of a performed system analysis to find the best system configuration (e.g., the system configuration with the minimum difference to the goal value). However, no guidance on how to choose the goal values is given and hence choosing the goal values in advance can be difficult. The subjective process of selecting the goal values will negatively influence the comparability and replicability of the results.

For the DEWI Security Metric a linear scale is used and some characteristic values are defined to support the process of choosing the goal values for each application scenario. For each of those values a certain security values is given and a description focusing on the attacker abilities and the required resources is provided. In Table 6.1 these characteristic values are summarized.

Privacy as part of security: in the SHIELD Multi-Metrics approach, security and privacy are treated as different aspects. However, in many applications and scenarios security and privacy issues are closely related to each other: all privacy- and security-related high-level requirements (nearly 100 requirements) collected within the DEWI project were mapped to technical security requirements in a detailed requirements consolidation phase.

Table 6.1: DEWI scale to define goal values: characteristic values and descriptions

Security level		Description
95	Very strong	Very hard for a team of experts with hardware above State of the Art (SotA) to break the system (upper practical limit)
75	Good	Experts need time AND access to the vehicle to break the system. No chance for non-experts
50	Average	Experts need time OR access to the vehicle to break the system. Non-experts need time AND access
25	Weak	Non-experts can break the system in affordable time OR with access to the vehicle
5	Very weak	Breaking the system with SotA hardware after a short web research possible for every one (lower practical limit)

Within the DEWI Security Metrics privacy is handled as part of security to decrease the complexity by removing one degree (i.e., privacy is not addressed individually anymore).

Evaluation of dependability: dependability is an integrating concept that encompasses the attributes availability (i.e., readiness for correct service), reliability (i.e., continuity of correct service), safety (i.e., absence of catastrophic consequences on the user and the environment), integrity (i.e., absence of improper system alterations), and maintainability (i.e., ability to undergo modifications and repairs) [4]. Rating system parameters w.r.t. this complex term is rather difficult as some of the aforementioned attributes can even be contrary: increasing the reliability and safety of a system (e.g., by maintaining a system every day) can decrease the availability of the system as it most likely cannot be used while being maintained.

Dependability is not supported within the current version of the DEWI Security Metric. Dedicated dependability metrics will be in focus of the DEWI successor project SCOTT[2].

Scope and terminology. Contrary to the SHIELD Multi-Metrics approach the DEWI Security Metric is currently only used to evaluate security (and privacy as part of security) aspects of a system, and dependability is out of scope for the analysis. In the following listing the terminology of the DEWI Security Metric is stated:

- *System*: the system or application (e.g., an automotive software update system), which will be analyzed by employing the DEWI Security Metric.
- *Subsystem*: a system consists of a set of subsystems (logical entities), which are interacting with each other. DEWI Security Metric allows nested levels of subsystems.
- *Components*: the smallest building blocks within the DEWI Security Metric. It is a logical unit (e.g., communication unit) assigned to a subsystem.
- *System parameter:* each component offers different parameters, which can be adjusted (e.g., communication unit as component; different encryption mechanisms as system parameters).

[2] EU ECSEL project Secure Connected Trustable Things (SCOTT). For further details see www.scottproject.eu.

- *Scenarios*: a system or application can be used in different scenarios (e.g., wireless software updates in a service center).
- *System configuration:* a set of system parameters. Varying these parameter values (e.g., using different encryption mechanisms in different configurations) will lead to several different system configurations.

Structured system analysis using the DEWI Security Metric. The DEWI Security Metric follows a structured approach encompassing six working steps. In the following these steps are described in more detail. Therefore, EASE-UP will be used to support the presented theoretical approach with a practical example. In [62] a comprehensive example on how to perform a security analysis for a WVI is presented and in [5] a complete security analysis of the developed local software update system using the DEWI Security Metric is described.

Step 1 – Define goal values for scenarios and different ECU types. In the first step the goal values are chosen for each scenario (i.e., software updates in the assembly line, during vehicle development, in the service center, as well as while the vehicle is parked at the smart home). The provided scale and guidelines (i.e., characteristic security values for specific attacker abilities and employed resources) will be used to find the required security level for each scenario.

The type of ECU should also be taken into account when employing the DEWI Security Metric for analyzing a wireless software update system, as discussed in more detail in [5]. Therefore ECUs are classified in three different classes [57]:

- *Uncritical - Class 1:* infotainment and entertainment related ECUs. Lowest criticality level as these ECUs have no impact on the safety of the vehicle.
- *Body and comfort - Class 2:* ECUs related to heating, ventilation, air conditioning, but also controlling the window lifters and windscreen wipers. Medium criticality level as this type have no direct impact on vehicle dynamics or the vehicle safety.
- *Powertrain, chassis, driver assistance - Class 3:* ECUs fulfilling complex and/or safety-critical tasks. Highest criticality level.

In Table 6.2 an example for goal values for all relevant local software update scenarios and the three classes of ECUs is provided. More information can be found in [57, 5].

Table 6.2: Security goals per local software update scenario and ECU class.

ECU class	Assembly Line	Vehicle Development	Service Center	Smart Home
Class 1	50	60	50	75
Class 2	55	60	60	80
Class 3	60	60	70	90

Step 2 – Decompose the system. Next, the system is divided into subsystems and components. Therefore, logical entities of the system are identified (e.g., single devices of a system such as the DT or the WVI) and then further divided into technical components (e.g., subsystem WVI: wireless communication unit as component). In Figure 6.1 a generic example of an decomposed system is shown.

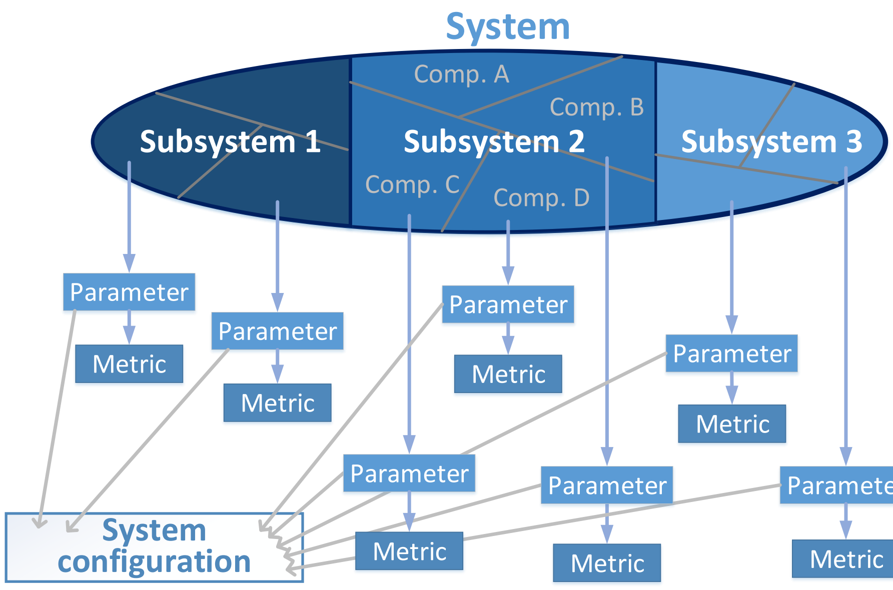


Figure 6.1: System decomposition. System to subsystem and finally components. System parameters are identified for all components and mapped to a specific configuration [5].

Step 3 – Identify the system parameters. In this step system parameters are identified for each component. Next, for each system parameter a metric including a criticality and a security value is set using the provided scale and the corresponding guidelines. This step is very important and requires expert knowledge. An example for a component and its system parameters is presented in Table 6.3.

Table 6.3: Example of a component and its system parameters. For each parameter the Security Value (SV) and the Criticality Value (CV) is provided.

Component/Parameter	SV	CV	Description
Key storage			Describes different ways of storing keys
Software	40	60	Stored in the software (e.g., #define) running
Memory (plain)	30	70	Stored in normal flash memory (plain)
Memory (encrypted)	75	25	Stored in normal flash memory (encrypted)
TPM	90	10	Trusted Platform Module (TPM) as key storage

Step 4 – Define suitable system configurations. To define a specific system configuration a system parameter is chosen for each component of the system (see also Figure 6.1). Thereby, one can create the most energy-efficient, cost-efficient, or strongest (w.r.t. security) system configuration by choosing the corresponding system parameters. Please note that there is no point in varying the system parameters in a way to get any possible configuration but one should only create practicable configurations.

Step 5 – Compute the security score per configuration. Once a system configuration is defined, the overall score of this configuration can be computed by applying the formulas defined in the underlying SHIELD Multi-Metrics approach (see [57] for more information on the formulas). Therefore, first the security score for each subsystem is computed and then these intermediate results are used to finally compute the overall system security score for the corresponding security configuration.

Step 6 – Find the most suitable system configuration per scenario. In the last step the results for all evaluated system configurations can be compared to the defined goal values to find the most suitable configuration for each considered scenario. Thereby, one can use the system configuration closest to the defined security score. But it is also possible to use the cheapest configuration fulfilling the security goal (i.e., management decision).

DEWI Security Metric supporting SAE J3061. The aforementioned steps allow to perform a structured system security analysis and are thereby also able to support the concept phase of the development cycle defined in SAE J3061 (see Section 2.4 for more information). Therefore, the goal value scale defined within the DEWI Security Metric can be mapped to the security goals described in the standard as shown in [62]. Furthermore, the results of a system analysis based on the DEWI Security Metric (i.e., the most suitable system configuration per scenario) can be used to simply perform the steps *Cybersecurity Concept* and *Identify Cybersecurity Requirements* as defined in SAE J3061: a system configuration can easily be transferred to a security concept as a configuration encompasses a suitable security mechanism (i.e., the chosen system parameter) for each technical component of a system. Furthermore, also security requirements can be easily extracted from a system configuration by converting a component (e.g., wireless communication unit) plus the corresponding system parameter (e.g., encryption mechanism A with key length X) to a textual requirement: The data transfer of the *wireless communication unit* shall be secured by *encryption mechanism A with key length X.*

Further information can also be found in [5] and [62].

6.1.2 Evaluation of the DEWI Security Metric

In this section the proposed DEWI Security Metric will be evaluated using an ordering relation. For this, the DEWI Security Metric will be employed to analyze a system (i.e., Wi-Fi protection mechanism), create system configurations according to established protection schemes (i.e., WEP, WPA, and WPA2), and finally compare the gathered results for each configuration. These results shall thereby confirm that the Wired Equivalent Privacy (WEP) mechanism provides the weakest protection, followed by Wi-Fi Protected Access (WPA) and WPA2 as the best option w.r.t. security according to [53]:

$$Security(WEP) < Security(WPA) < Security(WPA2) \tag{6.1}$$

To prove this hypothesis, the Wi-Fi protection mechanism and its main features are analyzed using the DEWI Security Metric in the following.

Generic Wi-Fi protection mechanism. In Figure 6.2 a generic Wi-Fi protection mechanism and its key features are sketched. Two main aspects are covered thereby. Connection-related security features are handling the establishment and maintenance of a trustable connection between an end device (e.g., a laptop or smartphone) and the Wi-Fi AP, while data-related security features ensure the confidentiality as well as the integrity of the exchanged data on packet level.

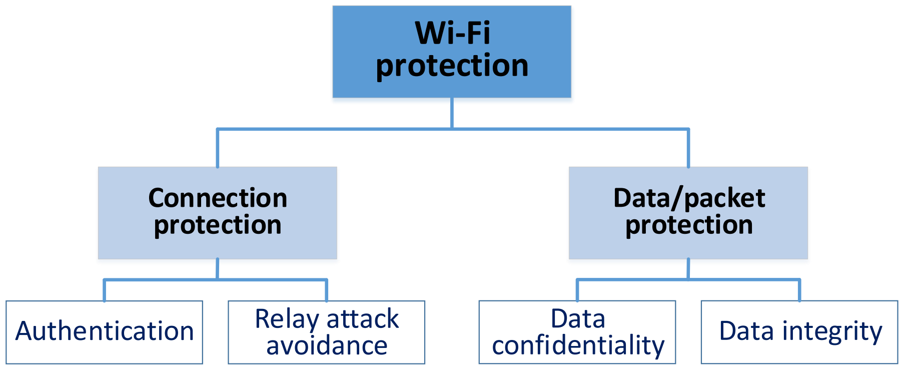


Figure 6.2: Generic Wi-Fi protection mechanism. Main features of a Wi-Fi protection mechanism encompassing connection- as well as data-related security measures.

The existing Wi-Fi protection mechanisms WEP, WPA, and WPA2 are also addressing these aspects and dedicate suitable protection measures to each of the required aforementioned security features. A summary of the employed protection measures employed by WEP, WPA and WPA2 is presented in Table 6.4.

Table 6.4: Different Wi-Fi protection mechanisms. Comparison of WEP, WPA and WPA2 [53, 54, 18] and there security features.

		WEP	**WPA**	**WPA2**
Data integrity	Type	CRC	MIC	MIC + CRC
	Length	32	64	64 + 32
	MIC key length	-	64	64
Data confidentiality	Type	RC4	RC4	AES
	Key length	40	128	128
Authentication	Type	-	4-way handshake	4-way handshake
	Key length	-	256	256
Relay attack avoidance	Type	-	inc. seq. number	packet number
	Key length	-	64	48

Decomposing the system. Figure 6.2 already provides a starting point for the decomposition of the system: one has to divide the system into subsystems and components. In Table 6.5 the results of the decomposition process are summarized and the weights on subsystem as well as on component level are stated. The two subsystems *Connection* and *Data* are thereby using the same weight as both aspects are equally important to ensure a trustworthy connection between an end device and the AP. The same is true for the components *Encryption* and *Integrity* of the subsystem *Data*. In the subsystem *Connection* the component *Authentication* was given a higher weight as this component has to mitigate several potential threats compared to the component *Relay Attack Avoidance* focused only on one specific type of attack.

Table 6.5: Decomposition of the Wi-Fi protection system into subsystems and components.

	Name	Weight
System	Wi-Fi protection	-
Subsystem I	Connection	50
Component I.a	Authentication	60
Component I.b	Relay attack avoidance	35
Subsystem II	Data	50
Component II.a	Confidentiality	50
Component II.b	Integrity	50

Creating system configurations for WEP, WPA, and WPA2. In the next step of the DEWI Security Metric approach the system parameters per component have to be identified and then suitable system configurations are defined. In the current example the system parameters are given by the analysed Wi-Fi protection mechanisms WEP, WPA and WPA2. The security features employed by these mechanisms are summarized in Table 6.4. These features can easily be transferred into system parameters for the components listed in Table 6.5 and then assigned to a system configuration. Table 6.6 shows the final system configuration for each Wi-Fi protection mechanism (i.e., WEP, WPA, and WPA2) as well as the corresponding security score per system parameter given in the brackets.

Use the DEWI Security Metric to calculate the security score. In a final step the DEWI Security Metric can be employed to calculate the security score for each system

Table 6.6: System configurations for WEP, WPA, and WPA2. Per configuration the identified system parameters plus the corresponding security scores (in brackets) are given.

Component	**WEP**	**WPA**	**WPA2**
Data integrity	CRC 32bit (40)	MIC 64bit length and 64bit key (75)	CRC 32bit plus MIC 64bit length and 64bit key (80)
Data confidentiality	RC4 with 40bit key (35)	RC4 with 128bit key (50)	AES with 128bit key (75)
Authentication	None (10)	4-way handshake using 256bit key (75)	4-way handshake using 256bit key (75)
Relay attack avoidance	None (10)	Incrementing sequence number with 64bit (50)	Packet number with 48bit (50)

configuration. Therefore the score is first computed on subsystem and finally on system level. The gathered results are presented in Table 6.7.

Table 6.7: The resulting security score on subsystem and finally system level (rounded).

Subsystem	Security score **WEP**	Security score **WPA**	Security score **WPA2**
Connection	10	67	67
Data	37	60	77
System	22	63	72

$$Security(WEP) = 22 < Security(WPA) = 63 < Security(WPA2) = 72 \tag{6.2}$$

The final security score values collected in Table 6.7 prove the defined hypothesis, conform to the security rating of WEP, WPA, and WPA2 as presented in [53, 54, 18], and show that the DEWI Security Metric leads to the expected results.

Sensitivity analysis. Next, the sensitivity of the DEWI security metric is discussed. This is important as different security experts using the DEWI security metric will potentially use slightly different values when performing the analysis, as the assignment of security score values per component is still based on a subjective judgment (although suitable scales have been defined to minimize the effect). The same is true for the chosen weight values on component as well as subsystem level. In the following, the robustness/sensitivity of the DEWI security metric is analyzed to prove that subjective judgment will not lead to significantly different results (i.e., the hypothesis is not valid any more). Please note that it is assumed that the analysis is performed by different security experts: although it is likely that slightly different values are chosen (e.g., as one expert is always using more pessimistic score values or lower weights in average), they all agree/know that one security mechanism (e.g., different encryption schemes) or parameter (e.g., the chosen key length) is more secure than another one. Cases where very diverse security scores are assigned for the same component will not be discussed in the following due to this assumption.

The following discussions require knowledge about the underlying equations of the DEWI security metrics (defined as part of the SHIELD multi-metrics; see also Section 2.5). Thus, these equations (see Equations 6.3 and 6.4) are briefly described in the following. More details can be found in [5, 21, 45].

Please note that the equations are not directly using the security value (S_i) of a component, but its criticality (C_i):

$$C = \sqrt{\sum_i \frac{c_i^2 * w_i}{\sum_i w_i}} \quad with \quad C = 100 - S, \tag{6.3}$$

$$w_i = \left(\frac{W_i}{max(W_i)}\right)^2 \quad and \quad S_i = 100 - C_i \tag{6.4}$$

with C being the resulting criticality value of the analyzed (sub)system, c_i the criticality of each of the components of the subsystem, W_i the weight values of the components (between 0 and 100), and w_i the normalized weight values (between 0 and 1).

Offset in weights: assigned weights W per component or subsystem have to be in a range of 0 (i.e., unimportant) to 100 (i.e., highly important). For two similar important subsystems, two experts could hence choose weight values such as *W_1=50 and W_2=50* or *W_1=70 and W_2=70*. In such a case, the DEWI security metric will lead to the same results, as the weight values are normalized (see equation in 6.4).

Different scales for weights: the DEWI security metric is also robust in cases where security experts use different scales for assigning weights (e.g., security expert 1 is assigning 25% higher weights than expert 2: *W_1=75 and W_2=50* for expert 1, and *W_1=60 and W_2=40* for expert 2): by normalizing the assigned weights the difference can be mitigated (i.e., in the given example the normalized weights will be w_1=1.0 and w_2=0.44 for both experts).

$$C = \sqrt{\frac{c_{I.a}^2 * w_{I.a} * w_I + c_{I.b}^2 * w_{I.b} * w_I}{(w_{I.a} + w_{I.b}) * (w_I + w_{II})} + \frac{c_{II.a}^2 * w_{II.a} * w_{II} + c_{II.b}^2 * w_{II.b} * w_{II}}{(w_{II.a} + w_{II.b}) * (w_I + w_{II})}} \tag{6.5}$$

Sensitivity w.r.t to security/criticality values: when performing a security analysis using the DEWI security metric, a security expert will assess all identified components of a system and assign security/criticality scores for each component accordingly. In case two experts are carrying out such an analysis, they will potentially use slightly different security score values. In contrast to the weights, where slight differences only cause minor variations w.r.t. the overall results of the analysis (or even lead to the same results as described earlier), differently chosen security/criticality values will have a more significant impact on the overall result. This is due to the fact that the square of the security/criticality value is used in the formula as shown in Equation 6.5, and hence a delta between two assigned security values will have a square impact on the result. Please note that in this equation, $c_{I.a}$ represents the first component (i.e., component *Authentication*) of subsystem 1 as presented in Table 6.5.

To support the security experts in the assessment process (i.e., choosing security values for each component) and, in further consequence, to minimize the impact of subjective judgment on the results of an analysis, a dedicated scale is provided as part of the DEWI security metric (please refer to Section 6.1.1 or [5, 57]).

6.2 Security Concept for Wireless Software Updates

The defined security design approach provides a structured methodology supporting the definition of a comprehensive security concept for automotive cyber-physical systems, and hence it was employed when defining the developed wireless automotive software update system. Detailed information about the performed analysis can be found in [57] and the results of the performed analysis can be found in [5] in its entirely.

In the following the defined security concept applicable for all addressed local software update scenarios (as described in Section 4.1) is described in detail. Therefore, first the identified security requirements (i.e., a result of the performed analysis) are summarized in Section 6.2.1 and the defined generic multi-layer security concept is presented in Section 6.2.2. Thereafter, in Section 6.2.3, the security features of the developed update protocol are described and the final hybrid security solution encompassing software as well as hardware security measures is proposed in Section 6.2.4. Finally, the fulfillment of the identified requirements is discussed in Section 6.2.5. A detailed formal security evaluation of the defined concept is described in Section 6.2.6.

6.2.1 Security Requirements for Wireless Software Updates

In the following the most important security requirements are stated and described in detail. These requirements are a result of the performed structured system analysis based on DEWI Security Metric: the resulting security configuration for EASE-UP can be easily transferred into security requirements by converting a component (e.g., wireless communication unit) plus the corresponding system parameter (e.g., encryption mechanism A with key length X) to a textual requirement. More information can be found in Section 6.1.1 (paragraph *DEWI Security Metric supporting SAE J3061*) as well as in [5].

REQ1 – Strong Authentication. Suitable authentication mechanisms must be employed to ensure that only i) authorized tools are allowed to connect to a vehicle (its WVI) to perform wireless diagnostics and software updates, and ii) authorized vehicles are receiving a new software update. The authentication step is essential for all addressed local software update scenarios.

REQ2 – Confidentiality of the exchanged data. The developed system and especially the utilized wireless network must ensure data confidentiality to protect the transferred software binary (relevant for all scenarios), the exchanged keys and other sensitive material (relevant for all scenarios), as well as user-related data to ensure the privacy of the involved users (especially relevant for service center and smart home scenarios).

REQ3 – Ensure data integrity. Data integrity is essential in all local software update scenarios and hence must be ensured by employing proper security mechanisms. The utilized mechanisms shall thereby protect against random bit errors while transferring a packet (e.g., due to interference) as well as ensure that attackers cannot tamper with the transferred data while transferred over a wireless network.

REQ4 – Key Storage. The security mechanisms employed to fulfill the aforementioned requirements REQ1 to REQ3 will demand several secret keys and other sensitive information. It is essential that this material is kept secret and cannot be revealed by an attacker. The developed system shall hence employ dedicated hardware such as a TPM to store secret keys and sensitive data.

REQ5 – Trusted core network. In local software update scenarios where a plug-in WVI is used to establish a connection between the vehicle and the local software update system (especially service center and vehicle development scenarios), single components (e.g., the WVI and/or a handheld) can potentially be used to perform unauthorized software updates or diagnostics (e.g., cash-in-hand jobs: a mechanic steals a WVI and performs illegal software updates and repairs). To avoid such a misuse of equipment, the developed system shall provide mechanisms to initially create a trusted core network involving the DT, the plug-in WVIs, as well as utilized handhelds. Once this trusted network was established, the WVIs and handhelds will only allow wireless diagnostics and software updates if they are connected to the trusted DT. Therefore, the equipment can only be used in close proximity (e.g., within a service center) to the DT. Please note that the DT is very likely kept in a restricted area or a dedicated room and only authorized users such as the head of a service center can access it.

REQ6 – Secured backbone link supporting vehicle authentication. In contrast to REQ 5, where plug-in WVIs as part of a trusted core network are employed to interconnect vehicles with the wireless software update system, REQ 6 is dedicated to scenarios where vehicles with integrated WVIs are involved. In such scenarios (e.g., a vehicle with integrated WVI is maintained within a service center for the first time), the WVI and the DT can potentially not establish a trustworthy connection without support of the OEM. Thus, the developed system must provide a suitable interface to the OEM backbone and be able to utilize this connection to request required information (e.g., the public key of the vehicle to verify its signature included in an authentication request sent by the vehicle).

REQ7 – User authentication and profiles. The wireless software update system must support dedicated user profiles to specify which group of users is allowed to perform a specific set of operations (e.g., a normal mechanic can only perform wireless diagnostics while a specially trained mechanic is also allowed to perform software updates). Additionally, the system shall provide suitable mechanisms to authenticate users when they utilize handhelds or directly interact with the DT.

REQ8 – Physical access to the vehicle cabin. A wireless software update needs to be authorized by proving physical access to the vehicle's inside (i.e., the cabin). This proof can be provided by i) connecting a plug-in WVI to the vehicle via its OBD interface which is located in the vehicle cabin (especially relevant for service centers), or by ii) pressing a dedicated button within the vehicle (mainly for the smart home use case). This security requirement is only relevant for the service center as well as the smart home scenario due to a required high security level (see Table 6.2).

REQ9 – Support secure multicast. The developed system must support trustworthy parallel software updates where several ECUs receive the same new software simultaneously (see Section 7.2.1 for more information). Thus, the wireless communication infrastructure as well as the developed update protocol must provide suitable mechanisms for secure multicast data streams.

6.2.2 Multi-layer Security Concept

The defined generic security concept encompasses security features on both the network as well as the application layer, as presented in Figure 6.3. On network layer the built-in security features of IEEE 802.11s are used. More information on these security features is presented in Section 5.1.4 and can also be found in [25].

The security features employed on application layer encompass i) a strong authentication scheme to set up a trustworthy connection between the DT, the WVIs and the handhelds, ii) mechanisms based on symmetric session keys protecting the integrity as well as confidentiality of the exchanged data (e.g., a software binary) thereby also supporting multicast data streams, iii) dedicated security hardware (i.e., TPM) to store keys and other sensitive material, iv) a user management scheme supporting user profiles as well as the user authentication on handheld devices as well as the DT, and v) out-of-band pairing features to create a trusted core network.

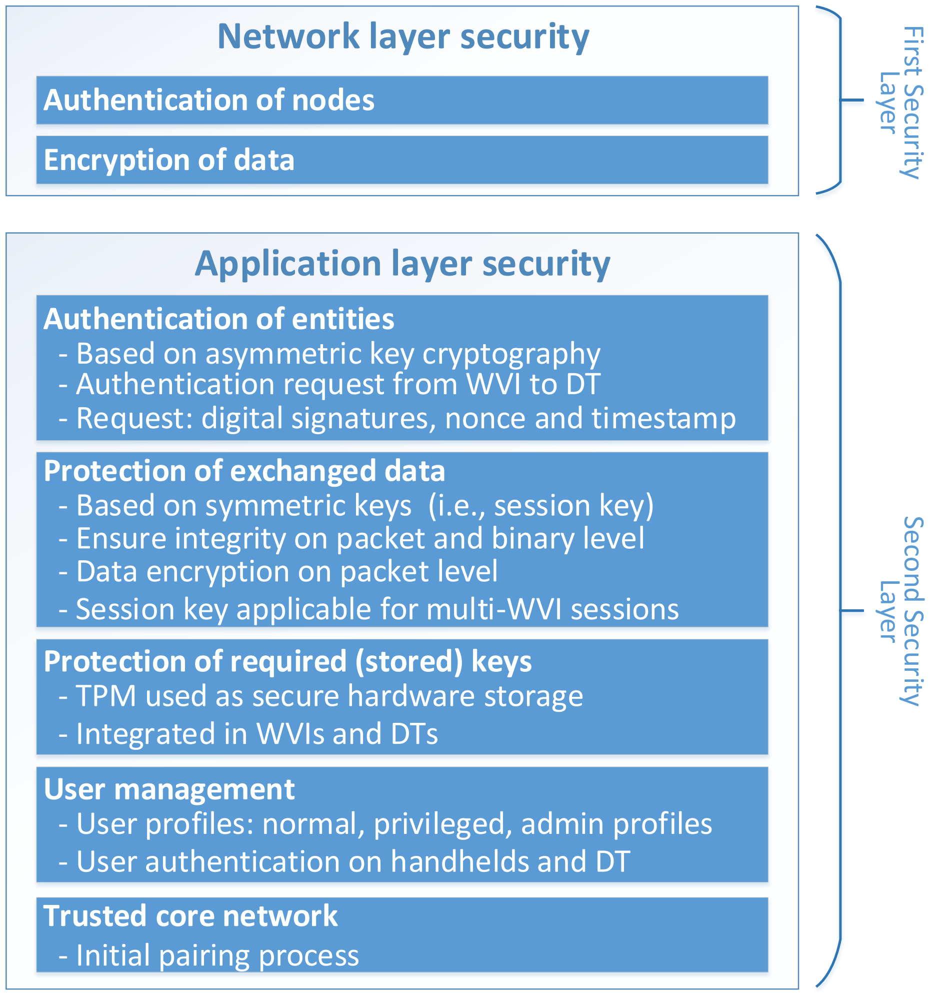

Network layer security
Authentication of nodes
Encryption of data
First Security Layer
Application layer security
Authentication of entities
- Based on asymmetric key cryptography
- Authentication request from WVI to DT
- Request: digital signatures, nonce and timestamp
Protection of exchanged data
- Based on symmetric keys (i.e., session key)
- Ensure integrity on packet and binary level
- Data encryption on packet level
- Session key applicable for multi-WVI sessions
Protection of required (stored) keys
- TPM used as secure hardware storage
- Integrated in WVIs and DTs
User management
- User profiles: normal, privileged, admin profiles
- User authentication on handhelds and DT
Trusted core network
- Initial pairing process
Second Security Layer

Figure 6.3: Generic security concept. The defined multi-layer security concept encompasses security features on network as well as on application layer.

6.2.3 Security Features of the Developed Update Protocol

The generic multi-layer security concept defines all relevant components required to secure a wireless automotive software update system. In this section, these aforementioned components are integrated in the defined software update protocol and technical details on the employed security features are stated. First, the utilized set of secret keys is discussed and thereafter the security features added to each of the six update steps of EASE-UP as defined in Section 5.2.1 are specified. An illustration of these security features is also provided in Figures 6.4 and 6.5.

Utilized security keys and key exchange. Several different keys are required by the employed security mechanisms. In the following these keys are discussed one by one.

Master key pair. The master key pair encompasses a private and a public key and is based on a RSA (i.e., public-key crypto system by Rivest, Shamir und Adleman). Thereby the developed system supports the use of 1024 and 2048 bit keys. Each device holds its own master key pair for its entire lifetime.

In a classic PKI the public keys are exchanged (e.g., over the Internet) and then used to encrypt data or to verify digital signatures. Thereby the problem of verifying that a public key really belongs to a certain user or entity arises, as attackers could potentially interfere this initial public key exchange and send there public keys instead. As a result, for two users (e.g., Alice and Bob) trying to exchange their public keys but receiving the public key of an attacker, no secure channel will exist as messages from Alice to Bob encrypted with the attacker's (however, Alice thinks its Bob's key) public key and signed with Alice's private key can only be encrypted with the attacker's private key (the same is true for encrypted messages sent from Bob to Alice). The attacker can also send signed messages to Alice acting as Bob (i.e., theft of identity).

To allow a user to verify that a public key really belongs to a certain entity or user, classic PKI systems employ a third party, the Certificate Authority (CA). The CA is a commonly trusted party maintaining digital certificates. In each certificate the identity of a user or an organization is linked to a public key. Furthermore, the certificate will also include other metadata such as an e-mail address or the URL of the organization's website, and specify a duration in which the certificate is valid. A user can hence request or verify a public key by requesting a certificate from a CA using an online request.

As a consequence, each node requires an Internet connection to communicate with a CA. In the defined security concept, a different approach has been chosen: to keep the system local (i.e., only the DT is connected to the Internet and/or the OEM backbone), a security concept without a CA was designed [37]. However, such a concept requires that public keys are initially exchanged and then securely stored. This pairing step is performed in a controlled environment (e.g., close proximity to the DT) using a dedicated media or mechanism (e.g., the developed system allows the use of NFC and one-time passwords) by authorized users (e.g., head of a service center). The initial pairing step (i.e., forming a trusted core network) only has to be done once: first, a master key pair is created on each device and securely stored (see REQ 4). Second, the public keys are exchanged within the system (e.g., between a DT and a handheld) and then securely stored. After the initial pairing step, the master keys can be used to handle the authentication between the involved nodes and, additionally, to sign as well as to encrypt unicast packets.

Session key. The symmetric session key is used to encrypt data exchanged between two entities within the developed local software update system (e.g., a DT and one WVI). The key is created by the DT for a bidirectional connection between a DT and a WVI or a handheld, or by the handheld for a connection between the WVI and the handheld. In the next step the key is encrypted using the public master key of the other entity (i.e., WVI or handheld) and signed by the private master key of the DT or the handheld, respectively. Finally the encrypted session key is sent to the WVI or handheld. All packets exchanged within the session key exchange handshake include timestamps and nonce to avoid replay attacks. Additionally, as each packet is signed using the private master key, also man-in-the-middle attacks are prevented.

A session key is typically valid for one continuous session. In the developed system the maximum session duration is 8h. If a session exceeds the 8h, a new session key is exchanged. Additionally, session timeouts (i.e., connection inactive for more than 1h) are defined. Both parameters (i.e., maximum session duration and session timeout) are configurable. The session keys are realized as Advanced Encryption Standard (AES) keys with a length of 128 or 256 bit.

Multicast-session key. This type of key is closely related to normal session keys despite the fact that a multicast-session key allows to send a secured multicast packet to several entities at the same time. The multicast-session key is created by the DT, then sent to several WVIs and finally used for all multicast data packets sent from the DT to these WVIs. Therefore, the DT once has to send an encrypted and signed packet including the multicast-session key to each WVI individually. The multicast-session key is again based on AES with either 128 or 256 bit length.

In the following the single steps of the software update protocol employed by EASE-UP are discussed w.r.t. the employed security features and the aforementioned security keys. An illustration of these security features is presented in Figure 6.4 and Figure 6.5.

Step 1 – Discovery and connection establishment: in this step the authentication between the involved entities (i.e., DT, handheld and WVI) is performed. Therefore the DT as well as the handheld will receive an authentication request from the WVI. This authentication request is signed by the private master key of the WVI. The DT will verify the request using the public key stored within its internal secure key storage and, if the signature is valid, create a session key. Next, the session key is encrypted using the public key of the WVI, signed by the private master key of the DT, and finally sent back to the WVI. The WVI will then verify the signature of the DT, decrypt the session key, and securely store it. All messages exchanged within this authentication handshake between the DT and the WVI (the same holds true for WVI to handheld as well as handheld to DT authentication) include nonce, identifiers, and timestamps to prevent replay attacks.

Step 2 – Gather vehicle information: no dedicated security features are required in this step. However, if parallel software updates are utilized within a specific scenario, this step can be used to i) identify if there is a potential for parallel updates (i.e., there are two or more vehicles demanding the same software update) and to ii) create a multi-cast session key and distribute it to all concerned WVIs.

Step 3 – Initialize a software update: this step requires to authorize the software update on ECU level using UDS between WVI and the ECU. For a majority of today's ECUs this step will be based on a Seed & Key mechanism, where the ECU creates a number of random bytes (i.e., the seed), sends the seed to the DT, and internally creates the key using a secret algorithm. The DT is also aware of the algorithm and can hence also compute the key and send it back to the ECU. Future ECUs will most likely employ stronger security mechanisms based on secure elements. In both cases (i.e., Seed & Key or authorization based on secure elements) the DT will be able to perform the authorization step by utilizing the information gathered by the vehicle's ODX file and/or the OEM backbone network.

In this step, the WVI will only act as gateway allowing the DT and the ECU to perform the authorization handshake. The WVI and the handheld will never hold algorithms or other information required in the authorization step due to security reasons.

Step 4 – Wireless data transfer: the wireless data transfer between the DT and the WVI is critical as attackers can try to eavesdrop or manipulate a software binary while it

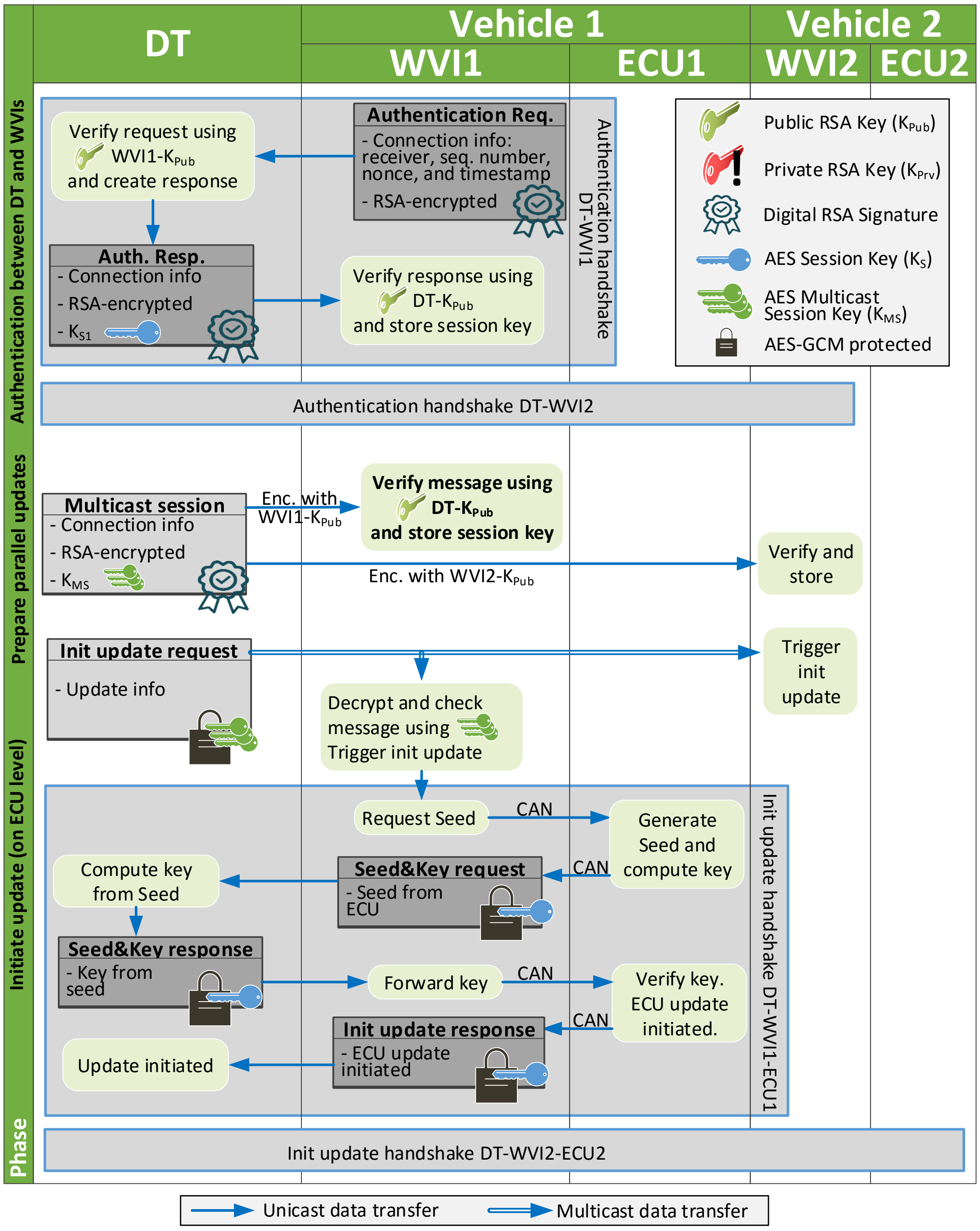


Figure 6.4: Securing EASE-UP: security features employed from *Step 1 – Discovery and connection establishment* to *Step 3 – Initialize a software update.*

is sent from the DT to the WVI. Thus, this step requires mechanisms to ensure data confidentiality as well as integrity. Both aspects are ensured by utilizing the current session key and by using AES in Galois/Counter Mode (GCM). AES-GCM allows AES-based data encryption plus provides stronger authentication assurance than a (non-cryptographic) checksum or error detecting code; in particular, GCM can detect both i) accidental modifications of the data and ii) intentional, unauthorized modifications [16]. Every data frame sent from the DT to the WVI is protected using AES-GCM. Additionally, once the entire binary was transferred to the WVI, the DT will create a hash of the binary using the Secure Hash Algorithm (SHA) SHA-256, sign it using its private master key, and send the singed hash to the WVI. The latter can use the hash to verify that the binary was transferred correctly in its entirely. In Figure 6.5 the security features of step 4 are illustrated.

In case of parallel software updates, the same approach is used: all data frames are protected by using the multicast-session key and by employing AES-GCM.

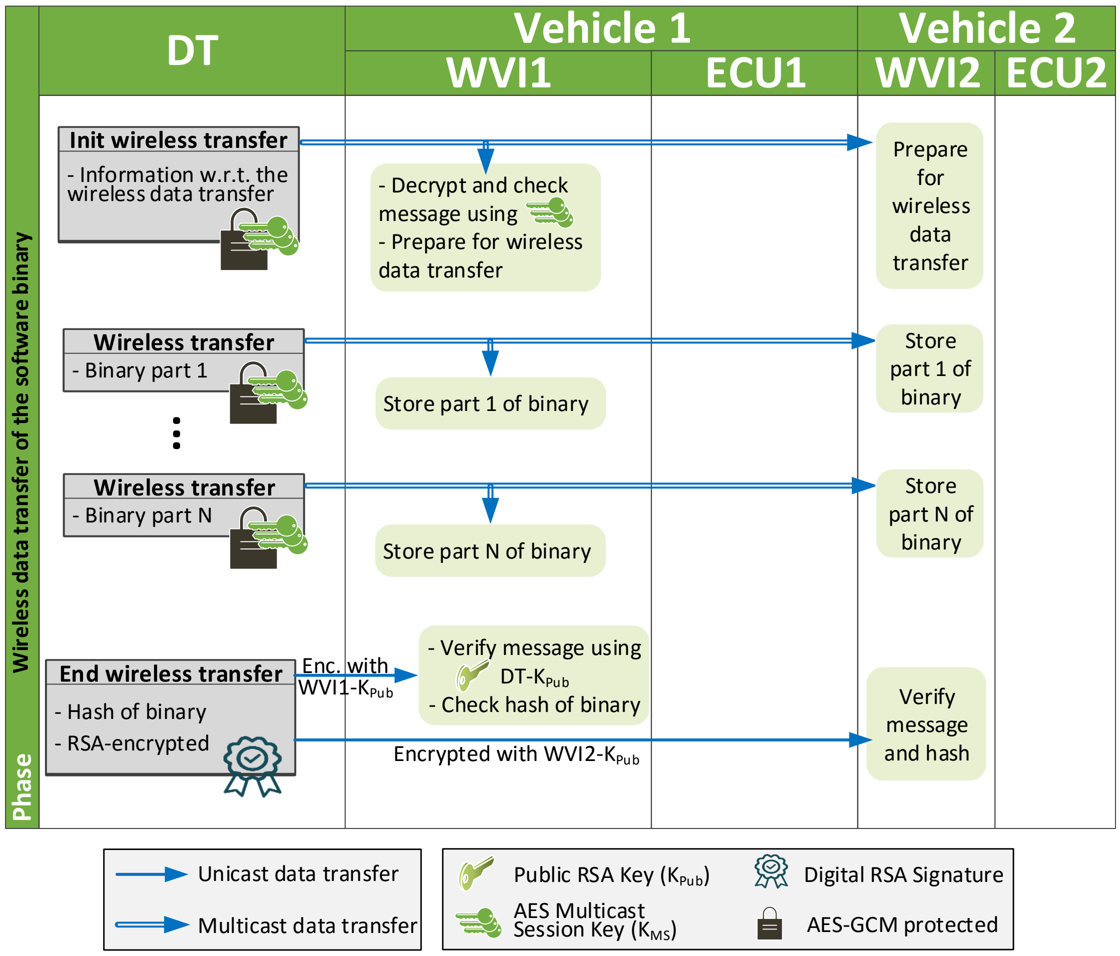


Figure 6.5: Securing EASE-UP: security features for *Step 4 – Wireless data transfer.*

Step 5 – Software download, and **Step 6 – Validation and reboot:** no dedicated security features are required for these two steps.

6.2.4 Hybrid Security Solution

The utilized TPM does not only provide a way to securely store secret keys and other sensitive material in hardware, but also allows to perform different cryptographic operations such as RSA encryption/decryption, creation and verifying digital RSA signatures, as well as AES data protections (e.g., AES-GCM). Thus, the TPM would be able to provide all suitable security measures (except SHA-256 support) required to protect the wireless software update protocol as described in the last section. However, it would also be possible to perform all required operations in software by utilizing suitable security libraries such as Java Bouncy Castle (JBC). To decide whether to perform all operations in hardware or software several experiments as presented in Section 7.2.3 and [37] were carried out. The performed measurements show that the TPM module is significantly slower compared to JBC, mainly due to a slow bus system interconnecting the security chip with the main controller. However, the TPM is required to securely store the required keys (especially the master key pair as well as public keys of other entities) and typically the private key cannot be extracted from the TPM once deployed (or the master key pair is even created by the TPM itself). This means that cryptographic operations where the private master key is required (i.e., the creation of a digital signatures as well as the decryption of RSA-encrypted data) must be performed by the TPM while other operations can be performed in software by employing the JBC.

The final security concept is hence built upon a hybrid solution utilizing hardware- as well as software-based security operations. This hybrid solution allows to combine the advantages of both, the efficient JBC library for data protection (i.e., AES-GCM-based security features) as well as the TPM for secure key storage and handling the authentication procedure.

6.2.5 Fulfillment of Security Requirements

In the last sections the defined security concept and the security extensions for the developed wireless automotive software update protocol were described. To evaluate the proposed security solution, first the fulfillment of all security requirement listed in Section 6.2.1 will be evaluated in this section and then a formal security concept evaluation is performed in Section 6.2.6.

REQ1 – Strong Authentication. RSA-based hardware-supported authentication is used between all involved entities. The authentication handshake is thereby also used to exchange the symmetric session keys.

REQ2 – Confidentiality of the exchanged data. Data exchanged between the involved entities (especially software binaries sent from the DT to the WVI) is encrypted using AES-GCM and either the session key (for bidirectional data exchange) or the multicast session key for multicast data streams (i.e., to perform parallel software updates).

REQ3 – Ensure data integrity. Data integrity is ensured by using AES-GCM. This protection mechanism prevents accidental modifications of the data as well as intentional, unauthorized modifications (i.e., tampering with the data).

REQ4 – Key Storage. Keys and other sensitive material can securely be stored on a TPM.

REQ5 – Trusted core network. A dedicated pairing mechanism was defined allowing to form a trusted core network. In this initial pairing step the public master keys are securely exchanged and stored.

REQ6 – Secured backbone link supporting vehicle authentication. Besides the possibility of forming trusted core networks, the generic security concept of the developed system also allows to involve the OEM backbone network in the authentication process (i.e., mainly relevant for fully integrated WVIs).

REQ7 – User authentication and profiles. A user management concept encompassing different user profiles as well as suitable user authentication mechanisms was developed. See [37, 57] for more information.

REQ8 – Physical access to the vehicle cabin. The proposed security concept can be configured in a way that the wireless software updates system requires physical access to the cabin of the vehicle as an additional authorization step (i.e., mainly relevant for service center and smart home scenarios).

REQ9 – Support secure multicast. The defined security concept as well as the developed software update system allow parallel software updates and provide suitable security mechanisms.

6.2.6 Formal Security Concept Evaluation

The defined security concept proposed in Section 6.2 is now formally evaluated using the Microsoft STRIDE methodology [48]. The latter can be employed to create a comprehensive threat model for complex systems. The resulting threat model can either be used to design a secure system or to evaluate an existing security concept.

In this section first general background information about the STRIDE methodology is given, and second the system is evaluated resulting in a list of potential system threats. These threats are then analyzed and used to verify that the proposed security concept encompasses suitable countermeasures mitigating the identified threats.

STRIDE threat model. The STRIDE threat model is an attack-centric approach allowing to analyze the security aspect of a system by identifying a number of potential security threats. These threats are thereby grouped into six categories: ***S***poofing identity, ***T***ampering with data, ***R***epudiation, ***I***nformation disclosure, ***D***enial of service, and ***E***levation of privilege.

Microsoft provides a free tool – the Threat Modeling Tool – allowing to model a system and to automatically extract the identified threats. The tool thereby also allows to prioritize the identified threats, to specify suitable threat mitigation measures and to keep track about the remaining threats.

A STRIDE-based security analysis is performed by analyzing each part of a system. Thereby every system component and process is evaluated w.r.t. all six threat classes. In the next step the identified threats can be sorted by its likelihood and impact.

Potential security threats. In the following the collected security threats are listed. Per STRIDE category, the most relevant threats (i.e., security threats with highest likelihood and impact) are stated and described. Further information can also be found in [37, 57].

Threat class 1 – Spoofing identity.

- Threat T1.1 – Spoofing the identity of a WVI: a malicious device acts as WVI to gather new software by acting as a vehicle with outdated software installed.
- Threat T1.2 – Spoofing the identity of a DT: a malicious device acts as DT to install malicious software on an ECU or to gather user-specific data from the vehicle.

Threat class 2 – Tampering with data.

- Threat T2.1 – Manipulate a software binary: an attacker tampers with the software binary while it is sent from the DT to a WVI to install a malicious software version on the ECU.
- Threat T2.2 – Tamper with an authentication handshake: an attacker manipulates an authentication handshake either to prevent a node (e.g., a WVI) to authenticate with another (e.g., a DT) or to gain access to the system itself.

Threat class 3 – Repudiation.

- Threat T3.1 – Performing an unauthorized software update (privileged user): a privileged user (e.g., a trained mechanic who is basically allowed to perform software updates) performs an unauthorized software update on a vehicle (e.g., the vehicle of a friend; without permission of the head of the service center).

Threat class 4 – Information disclosure.

- Threat T4.1 – Eavesdropping a software binary: an attacker eavesdrop the wireless channel while a software binary is transferred from the DT to the WVI. Thereby the attacker is able to gather the entire software binary.
- Threat T4.2 – Steal software update authorization keys: a malicious user (e.g., a mechanic) uses a handheld (i.e., an authenticated device) to eavesdrop the software update initialization process to gain access to the used authorization keys.

Threat class 5 – Denial of service.

- Threat T5.1 – Overflow the system with authentication requests: an attacker sends a vast number of authentication requests to the DT to overflow the system and hence to make the wireless software update system temporary unavailable.

Threat class 6 – Elevation of privilege.

- Threat T6.1 – Unauthorized user performing a software update: an unauthorized user (e.g., a mechanic without the required education) performs a software update by using the handheld device and/or the user account of an authorized/privileged user (e.g., a privileged and trained mechanic of the same service center).

Concept evaluation: security threats and countermeasures. Finally the identified security threats can be compared to the defined countermeasures. Therefore for each aforementioned threat the suitable countermeasure(s) are stated.

Threat T1.1 – Spoofing the identity of a WVI. For a successful attack, first the wireless network security (i.e., SAE features of IEEE 802.11s) must be overcome and second the digital signature of the WVI must be stolen. This can only be done by extracting the private key from the TPM of the DT, which is practically nearly impossible.

Threat T1.2 – Spoofing the identity of a DT. For this threat, similar considerations as for T1.1 apply. Additionally, the DT is often located in a restricted area and therefore accessing the TPM is even more challenging.

Threat T2.1 – Manipulate a software binary. AES-GCM is used to transfer a software binary from a DT to a WVI. Thereby each frame containing a chunk of the software binary is secured. Additionally, the hash value of the entire binary signed by the DT is sent to the WVI. An attacker would first have to gain access to the session key to successfully manipulate the AES-GCM protected data messages and second would need to forge the signature of the DT.

Threat T2.2 – Tamper with an authentication handshake. First, an attacker would have to overcome the security features of IEEE 802.11s. Then, to authenticate itself instead of the node trying to authenticate to a DT, the attacker would have to manipulate the authentication message. However, each message of the authentication handshake is protected by a digital signature and thus these type of attacks is mitigated.

An attacker could also try to keep another node from establishing a trusted connection to the DT by tampering with the authentication message in a way that the DT will not accept it. Although this attack only requires to overcome the security features of the wireless network (to identify authentication messages), it would require significant technical effort to constantly interfere with the authentication process and the impact on the system is rather small.

Threat T3.1 – Performing an unauthorized software update (privileged user). The developed system supports tracing of activities such as performing software updates. On the DT each performed software update will be logged together with the name of the mechanic performing the update as well as information about the vehicle. In a typical service center, the created log would reveal the unauthorized updates and the head of the service center could use the information to confront the user.

Threat T4.1 – Eavesdropping a software binary. Each data packet is encrypted using strong symmetric encryption based on AES. Successfully eavesdropping the wireless channel would thereby require equipment far beyond the current state-the-art.

Threat T4.2 – Steal software update authorization keys. The same security feature as explained in T4.1 are in place securing the messages required to authorize a software update. Furthermore, for Seed & Key or challenge-response authorization steps in general an attacker can only eavesdrop the solution for one specific challenge (i.e., the key for one seed) and thus cannot reuse this key later as the ECU will create a new challenge for each new authorization step.

Threat T5.1 – Overflow the system with authentication requests. In the current version of the developed system no dedicated protection mechanisms against Denial of Service (DoS) attacks are in place on application layer. However, an attacker would first need to break the security mechanisms on network layer before a DoS attack can be launched (e.g., by sending a vast number of authentication requests).

Threat T6.1 – Unauthorized user performing a software update. This type of threat is hard to prevent as it is typically performed by insiders: a privileged mechanic allows an untrained mechanic to use his personal user account and/or equipment to perform a software update (case 1) or the untrained mechanic uses the handheld device of a privileged colleague in an unnoticed moment (e.g., during a lunch break; case 2).

The threat connected to the latter case is mitigated by employing suitable timeouts: if the handheld device is inactive for a configurable duration (default value is five minutes), the user will need to re-authenticate with the device before it can be used again. Case 1 can only be detected by a superior. The developed system can support this by tracing all relevant user activities.

The performed formal analysis shows that all identified security threats (i.e., threats with the highest impact and likelihood) are prevented or at least significantly mitigated by suitable security mechanisms employed on network as well as application layer.

Chapter 7

Evaluating EASE-UP

EASE-UP, the proposed automotive software update framework described in Chapter 5, the designed security concept presented in Section 6.2, the employed wireless communication media, as well as the developed advanced software update mechanisms (Section 5.3) must be extensively tested to ensure that the software update process is performed in a reliable and secure way. This is especially important as EASE-UP allows to update all different types of ECUs including controllers performing safety-critical tasks in the vehicle. A system failure of such ECUs (e.g., due to a malfunctioning software version) can lead to severe accidents endangering the health and well-being of the driver, the vehicle passengers, as well as other road users.

Besides security and reliability, also related *efficiency* aspects must be analyzed to ensure that a wireless software update is performed in a secure, reliable but also fast way. Especially in highly automated environments such as a vehicle assembly line the efficiency aspect is of main importance as every delay (e.g., due to a slow software update) will interfere with the assembly process and thus can lead to significant extra costs.

In this chapter CESAR, a comprehensive testbed infrastructure for wireless automotive software updates is described and its key features are highlighted (Section 7.1). Thereafter, in Section 7.2, the results of different case studies carried out using CESAR are presented.

7.1 Testbed Infrastructure

In the following CESAR, a **C**onfigurable testbed infrastructure that allows to evaluate the effectiveness and **E**fficiency of wireless automotive **S**oftware updates in an **A**utomated and **R**epeatable way, is described in detail. First, some testbed requirements are discussed and second the developed, comprehensive testbed infrastructure is described [58].

7.1.1 Testbed Requirements

A proper testbed infrastructure should support the evaluation of the entire automotive software update process (i.e., six main steps as discussed in Section 5.2.1) and allow to study the impact of different aspects on its efficiency.

Supporting the entire update process. The entire software update process includes DTs, holding the new SW images as well as the required authorization keys, the WVI interconnecting the vehicle and the wireless infrastructure, an ECU where the new software image will be installed on (referred to as target ECU), and the network infrastructure (both wired and wireless links) interconnecting all these components. This implies that the nodes of a testbed should be able to support both DT and WVI roles. The WVI role requires to connect to one or more ECUs employing an automotive bus. This connection between a testbed node and one or several ECUs requires: i) the hardware to connect to the ECU using typical automotive bus systems (this encompasses bus controllers and the corresponding transceivers, physical interfaces, and additional hardware such as termination resistors), ii) the corresponding interface SW enabling the actual communication on top of the physical connection (e.g., CAN protocol), and iii) automotive diagnostic protocols on the application layer handling the data flow for larger frames and supporting different diagnostic functions (e.g., unlock the ECU, transfer data from/to the ECU). Furthermore, the testbed must support ECUs from different vendors and must hence provide standardized diagnostic protocol stacks on the application layer. The testbed shall also provide primary (but still exchangeable) target ECUs supporting standard-conform SW update mechanisms as well as allowing the utilization of special update mechanisms such as parallel and partial wireless SW updates.

Supporting different scenarios. The testbed should allow to emulate all relevant software update scenarios, including a large-scale assembly line scenario with up to 100 nodes, and support different wireless communication networks interconnecting these nodes. The testbed infrastructure shall hence allow multiple configuration profiles allowing the user to choose between different wireless network stacks (e.g., IEEE 802.11n or IEEE 802.11s) as well as network topologies (e.g., force or block routes), and provide configuration methods to influence single network links (e.g., increase/decrease the TX power). Furthermore, these configuration profiles shall include different security configurations on both the application as well as the network layer and allow to choose between different security parameters such as the authentication scheme (e.g., RSA-based) or the key length (e.g., 1024bit). This is important to analyze the impact of these security configurations on the duration of a software update and the efficiency of the entire update process.

Controlling experiments and logging. Ideally an extensive testbed should provide methods to (remotely) control the infrastructure and all involved nodes. These methods shall encompass mechanisms to start and terminate the experiments, to configure and reset testbed nodes, to request the current state and role of testbed nodes, and to collect the experimental results. The testbed shall also support the developer in measuring the SW update duration and create detailed timing profiles including the per-step latency of all required steps of the software update procedure. Furthermore, the testbed infrastructure shall provide configuration profiles allowing the fast and simple configuration of the testbed itself and all its nodes without a manual intervention of the developer.

Testbed deployment. The installation effort of the testbed shall be kept to a minimum and existing (network) infrastructures shall be reused whenever possible and applicable. As users typically act remotely and do not have physical access to the testbed nodes, the infrastructure should provide a way to power off and reboot each node remotely.

Remote control. The testbed should be remotely accessible from the Internet to allow users to monitor and control the status of the experiments. Nevertheless, the testbed network must be isolated from other corporate networks in order to minimize security breaches.

7.1.2 CESAR – Comprehensive Testbed Infrastructure

In the following an overview on CESAR, a comprehensive testbed infrastructure able to fulfill all the aforementioned requirements, is presented. Within this thesis the focus thereby is on the utilized testbed architecture. More details on the actual implementation of the testbed can be found in [58].

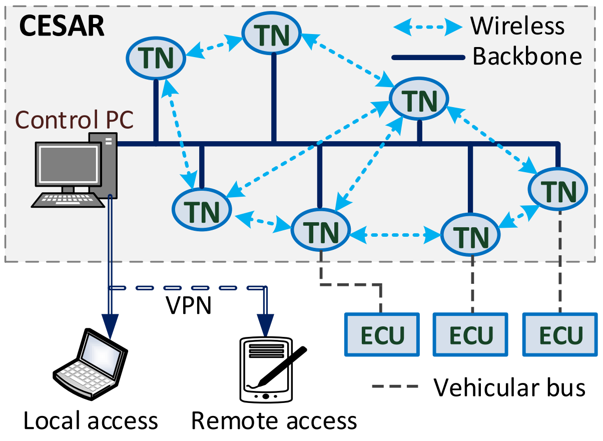


Figure 7.1: CESAR testbed architecture. TN are interconnected using a backbone network. A TN can be connected to one or several ECUs, or to real vehicles using automotive buses such as CAN.

In Figure 7.1 the designed architecture of CESAR is presented. CESAR basically consists of wireless TN and a Testbed Control PC (TC). The TNs and the TC are thereby interconnected by a wired back-channel. The TNs are utilizing a wireless network such as IEEE 802.11s or IEEE 802.11n to communicate with each other during the performed measurements while the testbed control as well as the collection of the gathered measurement data is done via the back-channel.

Testbed Control. The TC is controlling the entire testbed infrastructure and is responsible to collect and store the results gathered in the testbed. CESAR's TC provide a dedicated GUI allowing the user to start and stop experiments as well as to monitor the state of the testbed. The TC is also responsible to trigger the reconfiguration of the entire testbed by sending dedicated command messages to all concerned TNs via the back-channel.

Configuration profiles. CESAR is designed in a way that the configuration effort can be reduced to a bare minimum. Therefore, the concept of *configuration profiles* was introduced. A configuration profile is a set of configuration files and can contain i) specific security and/or network configurations, ii) a certain node occupancy allowing to emulate specific real-world SW update scenarios, iii) a set of system parameters such as the vehicle bus bit rate or the employed authentication mechanism, and iv) specific SW update mechanisms [58]. A developer can choose between different profiles, modify and merge them, and therefore easily switch between different experimental settings and redo a experiment later by selecting the corresponding configuration profile. In the latest version of CESAR, a configuration profile is a folder containing a set of configuration files (i.e., a text file with one configuration aspect per line). Default configurations for different testbed aspects such as predefined network topologies, sets of system parameters, or different logging mechanisms are provided as a starting point for developers. The developer can simply create a new profile by adapting these default configurations and pointing CESAR to the new profile (i.e., path of the folder). The management of the configuration profiles as well as the actual execution of them (i.e., the reconfiguration of the testbed according to the selected profile) is performed by the TC.

Testbed nodes. Each TN is configurable to assume different roles within CESAR. Depending on the selected configuration profile, a TN can act as a DT, WVI, or as relay node (i.e., only forwarding data in a multi-hop network such as IEEE 802.11s). Therefore, each TN hosts dedicated software implementations for each role running on top of a given hardware platform. The latter provides suitable automotive interfaces (i.e., bus systems such as LIN, CAN or FlexRay) to interconnect a TN to one or several target ECUs.

Target ECUs. One primary goal of the developed wireless software update system was to provide an OEM-independent solution. Therefore CESAR was designed to support different types of ECUs. This requirement is mainly achieved by employing automotive interfaces and connectors and by allowing different verification schemes. The latter is required to test if a new software binary was actually installed on the target ECU or not. Therefore, in the simplest case, one can implement target software binaries which are periodically sending a software version ID on the automotive bus. For example, for a specific target ECU two software binaries A and B exist. If binary A is installed, the ECU will send a CAN frame with payload 0x1 every second on the CAN bus while binary B will broadcast a frame with 0x2 per second. Besides this simple but efficient verification mechanism, the main target ECU employed within CESAR also allows more sophisticated checks. Therefore, the hash of the entire ECU memory is computed after a software update and in further consequence used to verify that the binary was successfully (i.e., completely and error-free) installed. The main target ECU additionally allows CESAR to monitor the current state of the ECU (via CAN) while a software update is performed. This feature is realized by sending specific CAN requests (i.e., frames with predefined IDs) from the TN to the connected ECU.

Remote control. CESAR allows to control the testbed remotely by employing the TC as a secure gateway between the Internet and the testbed infrastructure (i.e., the TNs interconnected by the back-channel). This remote access can be used to i) monitor the state of the current experiment and/or the testbed, ii) start and stop experiments, as well as iii) re-configure the testbed by selecting a different configuration profile.

Prototype testing. In the developed testbed infrastructure the performance of different software versions can be compared in a highly automated manner. Therefore, CESAR stores and maintains all developed software prototypes in a centralized repository allowing a user to select a certain software version by choosing a specific configuration profile.

7.2 Efficiency Evaluation – Case Studies

CESAR was used to carry out different case studies to i) evaluate the performance of different software update mechanisms as presented in Section 7.2.1, ii) analyze the applicability of different wireless protocols (i.e., IEEE 802.11n and IEEE 802.11s) for local software update as well as to test the latest implementation of IEEE 802.11s (i.e., open11s [11]; results are described in Section 7.2.2), and iii) evaluate the impact of different security configurations on the efficiency of the entire system as discussed in Section 7.2.3.

7.2.1 Performance of Software Update Mechanisms

In this section the performances of different software update mechanisms are compared with each other. Therefore a software update of the main target ECU is performed twenty times using a specific test software with a size of 445 KB for each of the tested update mechanisms, namely

- *Basic* wireless software update;
- *Partial* wireless software update;
- *Parallel* wireless software update.

As a baseline the aforementioned test software is installed on the main target ECU using a traditional, UDS-based wired update mechanism. The gathered results will be briefly discussed in the following. More detailed information on the performed experiments and the collected results can be found in [37, 58, 61].

Profiling the software update duration. CESAR allows to create detailed timing profiles for a wireless software update. In particular, one can not only measure the overall duration of the software update process but it is also possible to analyze the latency added by each of the six steps defined within the designed wireless software update protocol (see Section 5.2.1 for more details). Please note that *Step 2 – Gather vehicle information* was not considered in the following evaluations and timing profiles as the update was only performed for one specific ECU (i.e., Volvo FlexECU) and thus no information about the vehicle was collected nor required (see also [37]).

In Figure 7.2 an example of a measured timing profile is presented. *Step 5 – Software download* is responsible for a large share (i.e., 64.0%) of the overall duration due to the slow CAN bus employed between the ECU and the WVI. *Step 1 – Discovery and connection establishment* including the RSA-based authentication between the entities adds about 11%, and *Step 3 – Initialize a software update* about 1% to the overall duration. The second largest share is added by *Step 4 – Wireless data transfer*. This step is responsible for about 21.6% of the time required to update the software of an ECU. The last step, *Step 6 – Validation and reboot*, represents about 2.4% of the duration of an update.

Overhead of wireless software updates. A traditional wired software update encompasses the minimum set of required steps to install new software on an ECU: initialize and authorize the software update on ECU level, download the new software to the ECU over an automotive bus (i.e., most likely CAN), and reboot the ECU. In comparison, wireless

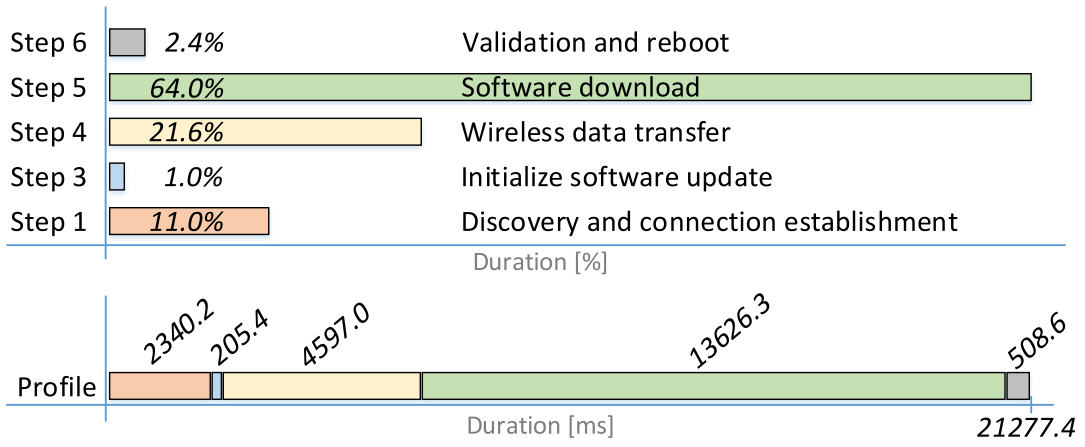


Figure 7.2: Timing profile of a wireless software update. Latency added by each step leading to the overall update duration.

software updates require some steps and hence also add some overhead to the overall software update duration. Please note that i) in this comparison some manually performed steps (e.g., by the mechanic) such as interconnecting the DT with the vehicle using a dedicated wire are not considered and ii) the wired mechanism is used as benchmark where only the overall update duration is provided but no information about the per-step latency is available.

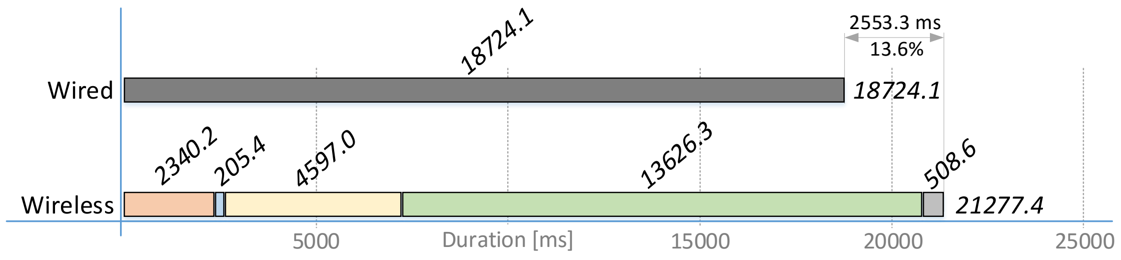


Figure 7.3: Comparison of wired and wireless software update mechanisms. The overall duration of a wired as well as a wireless software update is measured and the overhead added by the basic wireless software update protocol is analyzed.

In Figure 7.3 the duration of the wired and the wireless update mechanism is shown and the overhead added by the wireless protocol is highlighted. This overhead is mainly due to additional latency added by network establishment mechanisms and the employed security measures. Although the overall duration is slightly (13.6%) increased, the basic wireless software update can still be very beneficial in different local software update scenarios as it allows to connect to different vehicles and ECUs seamlessly (i.e., without disconnecting from one vehicle to connect to another). Thereby, a user (e.g., a mechanic in a service center) can work on several vehicles in parallel: the mechanic can diagnose and repair vehicle A while vehicles B and C receive new software for their ECUs.

Evaluation of advanced update mechanisms. To actually decrease the overall duration of a wireless software update and thereby to increase the efficiency of the system, two different advanced update mechanisms – partial and parallel wireless software updates – were developed (see Section 5.3) and will be evaluated in the following.

Partial software updates. One approach to actually decrease the time required to install new software is to reduce the number of bytes to be transferred. This is especially relevant as the data download over CAN takes a significant portion of the overall update duration as shown in Figure 7.2. Partial software updates allow to transfer only parts (e.g., one code segment instead of the entire binary). In the following experiment only one code segment with a size of 1 KB has to be transferred to the ECU (i.e., the main target ECU) instead of the entire software binary with a total size of 445 KB.

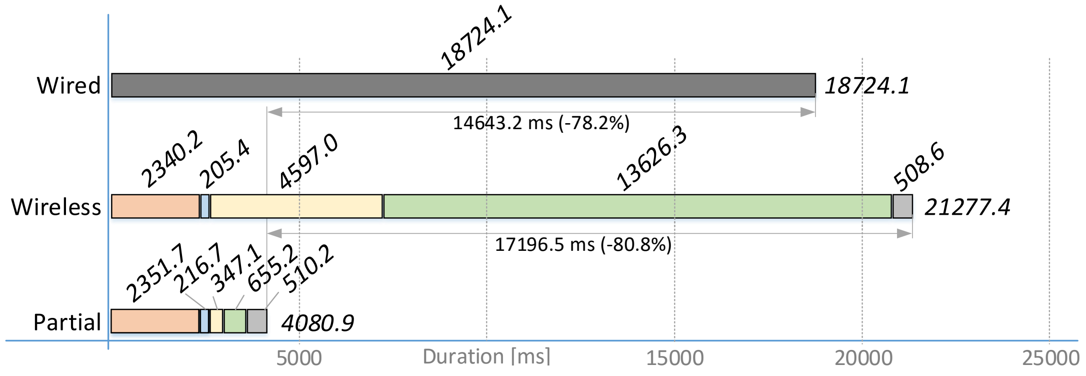


Figure 7.4: Benefits of employing partial software updates. The performance of a partial software mechanism is compared to a traditional wired update as well as to an update performed using the designed basic software update protocol.

The results of this experiment are shown in Figure 7.4. As expected, the overall duration was significantly reduced (i.e., a duration decrease of 83%). With about 58% of the overall duration, the largest share is now added by *Step 1 – Discovery and connection establishment* and thereby especially the RSA-based authentication process. In comparison, the CAN-based data download from the WVI to the ECU is now only responsible for about 16.1% of the overall time required to perform the update.

A reduction of the data to be transferred can significantly speed up the software update process. However, the actual benefit is really depending on the size of the original binary, the size of the code segments which are transferred to the ECU (instead if the entire binary), and the automotive bus system interconnecting the WVI and the ECU. Furthermore, also the segmentation of the binary itself significantly influences the potential benefits. Please also refer to the *Guidelines for efficient partial software updates* discussed in Section 5.3.

Parallel software updates. Parallel software updates are targeting situations, where the same new software needs to be installed on several vehicles. Instead of installing the new software on the vehicles sequentially (i.e., one by one) as it would be the case for wired updates, the developed parallel update mechanism can be employed to install the software on several vehicles/ECUs simultaneously. This parallel approach can thereby help to decrease the overall duration (i.e., time required to install the software on all vehicles) significantly.

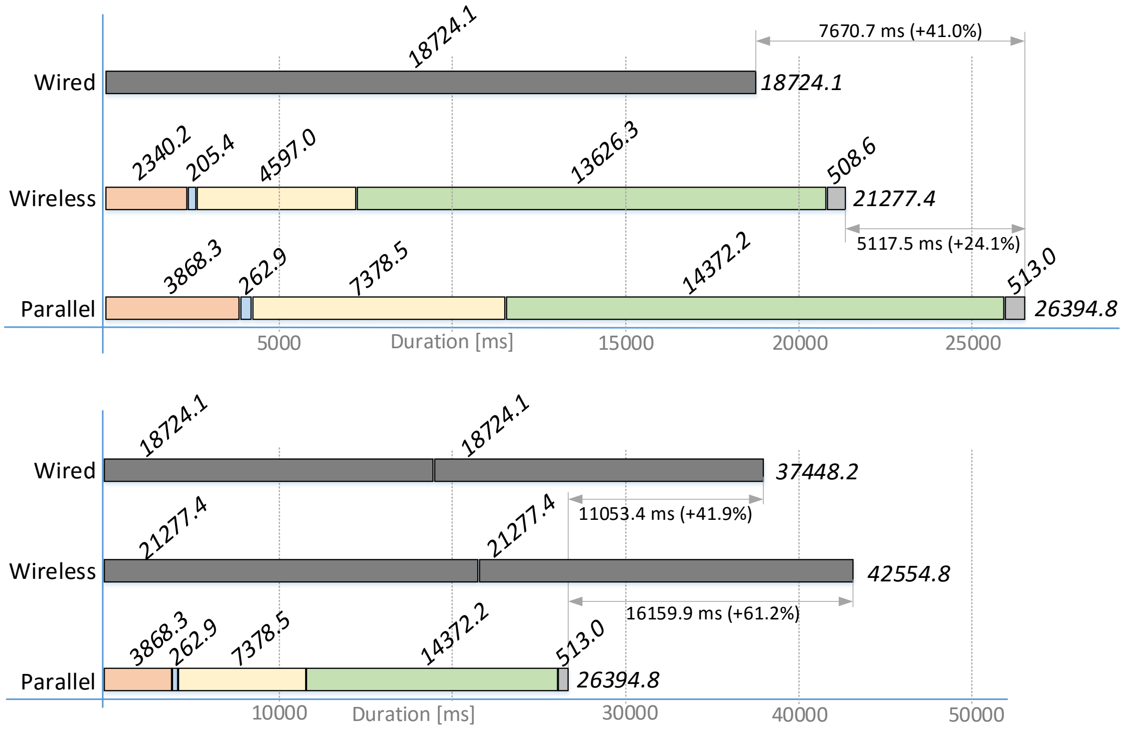


Figure 7.5: Overhead (top) and benefits (bottom) of employing parallel software updates. Two software updates are done in parallel instead of performing them sequentially.

In Figure 7.5 evaluation results are shown. Thereby, the duration of a parallel software update for two ECUs is compared to the time required to install the software sequentially on both ECUs using i) wired and ii) wireless (i.e., the designed basic update protocol) approaches. The evaluation results presented in Figure 7.5 show that the parallel update duration is increased by about 24% compared to a normal wireless software update and about 41% compared to a wired update. This overhead is due to the fact, that the required steps cannot by parallelized completely. However, parallel software updates are still way faster compared to performing software updates sequentially (i.e., performing an update two times in a row): the duration is decreased by about 61% compared to wireless and about 42% compared to wired software updates, respectively. Thus, parallel software updates can significantly help to improve the efficiency of the developed software update system. This is especially true for situations, where several vehicles receive the same software update.

Summary of the analyzed update mechanisms. Different wireless software updates are supported by the developed wireless software update framework. Each of these mechanisms was evaluated and compared to the performance of a baseline system based on traditional wired software updates. The gathered results are summarized in Table 7.1. The developed framework provides a significantly more flexible approach to perform wireless diagnostics and software updates, while keeping the added overhead (w.r.t. the update duration) at a minimum. Furthermore, the developed advanced software update mechanisms are able to significantly reduce the time required to perform wireless software updates compared to the traditional wired solution. Appropriate security mechanisms provide suitable mechanisms protecting all involved nodes, the connected vehicles, as well as the exchanged data while still allowing an efficient software update process.

Table 7.1: Summary of collected evaluation results. Comparison of different wireless software update mechanisms and the wired benchmark baseline.

Mode	**Step 1**	**Step 3**	**Step 4**	**Step 5**	**Step 6**	**Total**
Wired	Information not available for wired updates.					18724.1 ms
Basic wireless	2340.2 ms (11.0%)	205.4 ms (1.0%)	4597.0 ms (21.6%)	13626.3 ms (64.0%)	508.6 ms (2.4%)	21277.4 ms
Partial	2351.7 ms (57.6%)	216.7 ms (5.3%)	347.1 ms (8.5%)	655.2 ms 16.1%)	510.2 ms (12.5%)	4080.9 ms
Parallel (2 ECUs)	3868.3 ms (14.7%)	622.9 ms (1.0%)	7378.5 ms (28.0%)	14372.2 ms (54.5%)	513.0 ms (1.9%)	26394.8 ms

7.2.2 Network-related Evaluations

Besides evaluating and comparing different wireless software update mechanisms, CESAR also allows to analyze the impact of the employed wireless network on the efficiency of wireless software updates. Furthermore, the developed testbed infrastructure i) allows to test actual implementations of these protocols (e.g., open11s as the most advanced implementation of IEEE 802.11s), ii) can be used to verify that the network is working as expected, and iii) helps to identify any network-related issues.

Wireless media for local updates – IEEE 802.11n vs. IEEE 802.11s. The choice of a wireless media employed to perform a wireless software update will significantly impact the dependability of the entire system. Because of that, different wireless protocols were already discussed in Section 5.1, and their key features w.r.t. wireless software updates were discussed. As a result of this comparison, IEEE 802.11s was identified as the most suitable candidate. Besides IEEE 802.11s, also IEEE 802.11n can be an interesting alternative mainly due to its high bandwidth and strong security features.

In the following, IEEE 802.11n and IEEE 802.11s are employed as wireless media in an emulated service center scenario, where several vehicles within a service center shall receive new software. The scenario is sketched in Figure 7.6. Node 9 is acting as DT and performs software updates on the ECUs connected to nodes 1, 2, 4, and 6 (these nodes are acting as WVIs). For the experiment, ten wireless software updates are performed per ECU and wireless media (i.e., IEEE 802.11n and IEEE 802.11s) [58].

It is important to highlight that, in the chosen configuration, the links between node 9 and the other TNs are of different quality due to the placement of the nodes and the resulting distance between them. As shown in Figure 7.6, node 1 is the furthest away (from node 9 point of view), and is not in the direct communication range of node 9. As a result, no communication can be established between these two nodes when employing IEEE 802.11n. In contrast, when using IEEE 802.11s, the nodes can exchange data by employing multi-hop routes. Therefore, data packets will hop through additional nodes and thereby only use very good links to maximize the reliability of the communication.

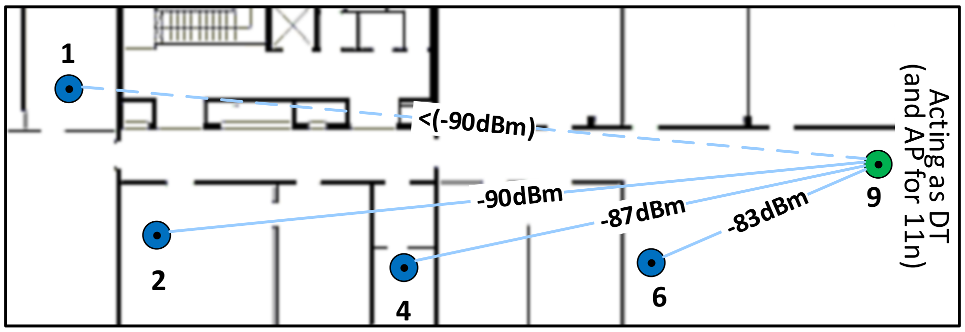


Figure 7.6: Wireless software updates using IEEE 802.11n and IEEE 802.11s and comparing the performance using an emulated service center scenario in CESAR. Node 9 is configured to work as DT, and also as AP when using IEEE 802.11n. Nodes 1, 2, 4, and 6 are acting as WVIs and are connected to ECUs.

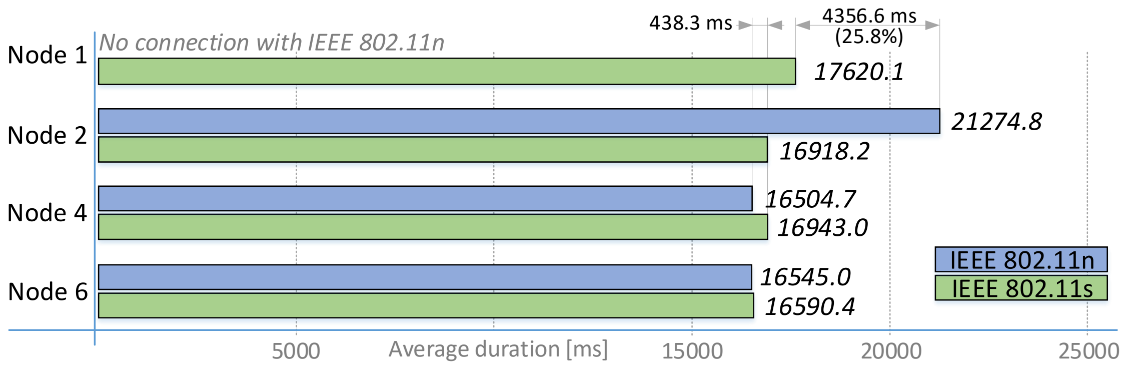


Figure 7.7: Software update duration for updates performed using IEEE 802.11n and IEEE 802.11s. The ECU connected to node 1 cannot be update when using IEEE 802.11n.

The gathered results are presented in Figure 7.7. For good links (i.e., between node 9 and nodes 4 and 6), both protocols nearly exhibit the same performance [58]. For node 4 IEEE 802.11n, in average, is about 400 ms faster than IEEE 802.11s. This slightly increased packet latency is due to the fact that some multi-hop paths were used by IEEE 802.11s as a result of lost packets on the direct link: the detailed results gathered using CESAR show an average path length of 1.06 hops. In case of intermediate links such as between node 2 and node 9, IEEE 802.11s outperforms IEEE 802.11n by a factor of about 25%. For node 1, only IEEE 802.11s is able to allow a wireless software update due to its multi-hop ability.

The performed experiment shows that IEEE 802.11s is indeed the right choice as it offers similar high bandwidth as IEEE 802.11n while ensuring reliable data exchange even in challenging situations and environments (for wireless communication).

Isolate node problem and its solution. CESAR can also be used to investigate the connectivity of wireless nodes and to expose issues in a network protocol implementation. In this case study a scalability issue of the default open11s [11] implementation is revealed

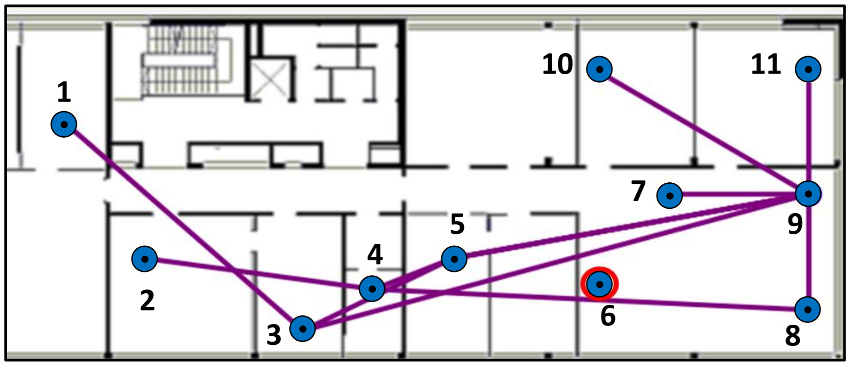


Figure 7.8: Screenshot showing the actual deployment of the TNs. Purple lines are representing the established connection tree with node 9 as root. All other nodes, except node 6, are connected to node 9 (either directly or via other nodes in between).

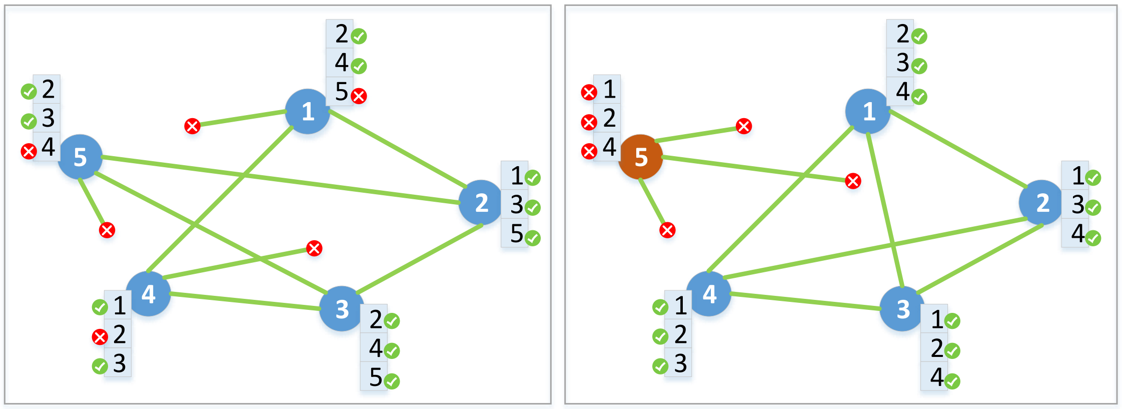


Figure 7.9: Peer links between nodes in an IEEE 802.11s network are very dependent on the sequence the nodes join. Left: an inefficient structure as node 5 is only connected to far away nodes. Right: node 5 is isolated from the rest of the network.

and a solution to mitigate this issue is proposed.

The scalability issue was first observed when configuring CESAR to use IEEE 802.11s as wireless communication media for an update scenario encompassing 11 wireless nodes: some nodes were not reachable by other nodes of the network, despite being physically close to each other. An example of this problem, from now on referred to as the *isolated node problem*, is depicted in Figure 7.8. This figure shows a screenshot of the testbed user interface displaying a snapshot of the connectivity in the network at a given point in time [58]. The figure shows a connection tree (purple lines) with node 9 as root. The tree shows that all other nodes, except one (node 6), are either directly connected to the root node or via several hops. Node 6, marked with a red circle, however, is isolated from the network and cannot exchange data packets with any other node. The observed scalability issue is critical as SW update scenarios can encompass several vehicles in a dynamic and changing environment where vehicles are frequently joining and leaving the network.

To shed light on this problem, CESAR was used to carry out several connectivity tests (i.e., different sequences of joining nodes) and thereby to collect link and path information. The performed experiments reveal two major problems of the default open11s implementation caused by its limited neighbor table size. The employed TNs use Wi-Fi sticks allowing to store up to seven (CESAR allows to configure different limits) established peer links. Once the maximum number of links is reached new requests will be declined. In Figure 7.9 the problem is illustrated using 5 nodes with a neighbor table size of three (i.e., each node can store information about up to three neighbors).

Inefficient network structure. In IEEE 802.11s peer links are established between two nodes if i) the nodes can *hear* each other and ii) the nodes have a free spot in their neighbor table. Hence, the link structure of an IEEE 802.11s network is mainly influenced by the sequence of nodes joining the network (and not if a node is close or far away). This fact leads to inefficient links affecting the network performance, as shown in the left illustration of Fig. 7.9.

Isolated node. In the worst case the limited neighbor table size can even lead to isolated nodes (Fig. 7.9, right illustration): nodes 1 to 4 have successfully established stable peer links with each other (e.g., as all nodes join the network at the same time). Node 5 (e.g., joining later) will send peer-link-establishment requests to nodes 1 to 4 to properly join the network. However, as nodes 1 to 4 already have three neighbor stored in their neighbor table, they will decline these requests and node 5 will be isolated from the network.

This identified issue was solved by adapting the latest `open11s` implementation: i) adding new messages to inform the network about isolated nodes, and ii) implementing algorithms to solve the *isolated node problem*. The adapted open11s version can be chosen and configured by CESAR besides the default open11s version.

7.2.3 Impact of Security Mechanisms on Efficiency

The employed security mechanisms are required to protect the exchanged data as well as all involved entities and users. However, these mechanisms also impact the efficiency of the system and add additional latency to some of the steps of the defined update protocol. In the following some interesting security evaluations are presented and the impact on the efficiency of the developed update system of different security features is analyzed.

Impact of IEEE 802.11s security features on network layer efficiency. In IEEE 802.11s SAE [25] is utilized to protect the transferred data (see also Section 5.1.4). In open11s, the SAE security features can be disabled to set up an open (i.e., without any security features; no password required to join the network) wireless network. In the following this ability is exploited to evaluate the impact of SAE on the network layer performance.

The experiment performed within CESAR measures the Round Trip Time (RTT) for different multi-hop paths with SAE enabled/disabled. The configured, static multi-hop routes (i.e., the routes are forced and the routing algorithm is disabled) between node A to node B encompass 1-hop, 2-hop, 3-hop and 4-hop paths as sketched in Figure 7.10. Relay nodes R1-R3 are used to forward data packets according to the defined routes.

For each defined route 1000 UDP packets were sent from node A to node B and back and the RTT was measured. The gathered results are presented in Figure 7.11. The green, continuous line shows the RTT for different numbers of hops with SAE on and the

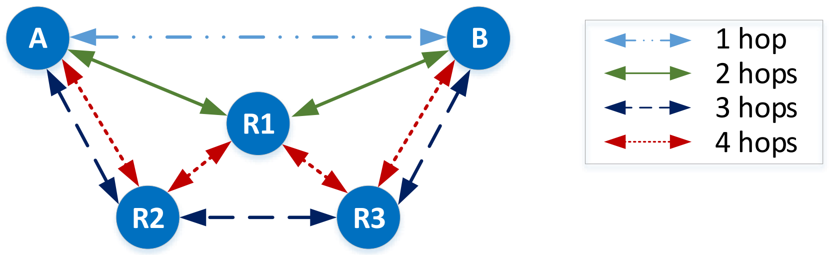


Figure 7.10: Static multi-hop routes between node A and node B using relay nodes R1-R3.

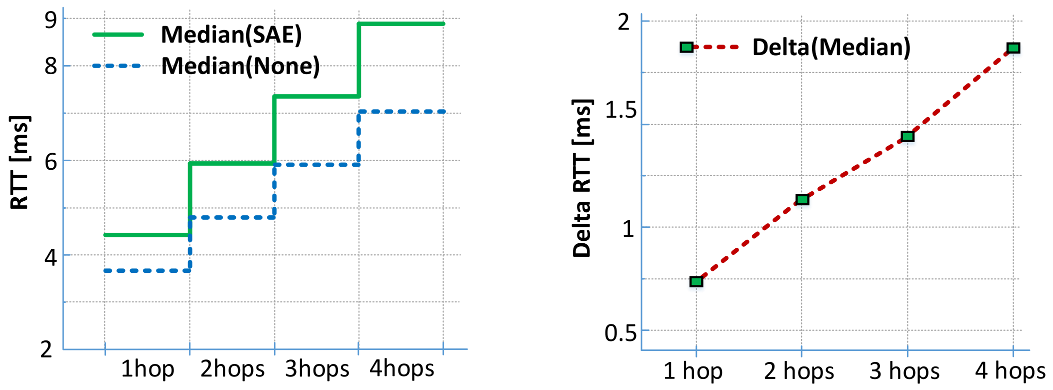


Figure 7.11: Impact of SAE on the network efficiency. SAE security features enabled vs. SAE disabled (i.e., None). Median (left), and delta(median)=median(SAE)-median(None) (right) of the RTT measurements using 10000 UDP packets are shown.

blue, dashed line for the measured RTT with SAE disabled. 7.11 shows that each hop significantly increases the RTT (see also [58]). This is due to the fact that in SAE a packet has to be encrypted and decrypted at each hop in between node A and node B.

Impact of different security configurations. In addition to the employed security features on network layer (i.e., SAE), EASE-UP also utilizes strong security features on application layer. In the following, the impact of all employed security features on the overall performance is evaluated by measuring the software update duration (445 KB binary) for different security configurations using CESAR and the Volvo FlexECU as target ECU. Four different configurations were tested:

- Configuration C1 (baseline): all security features disabled;
- Configuration C2: only security features on network layer;
- Configuration C3: only security features on application layer;
- Configuration C4: all security features enabled.

The gathered results, presented in Table 7.2 as well as Figure 7.12, are showing that the employed security mechanisms increase the overall update duration by 18.5%. Although

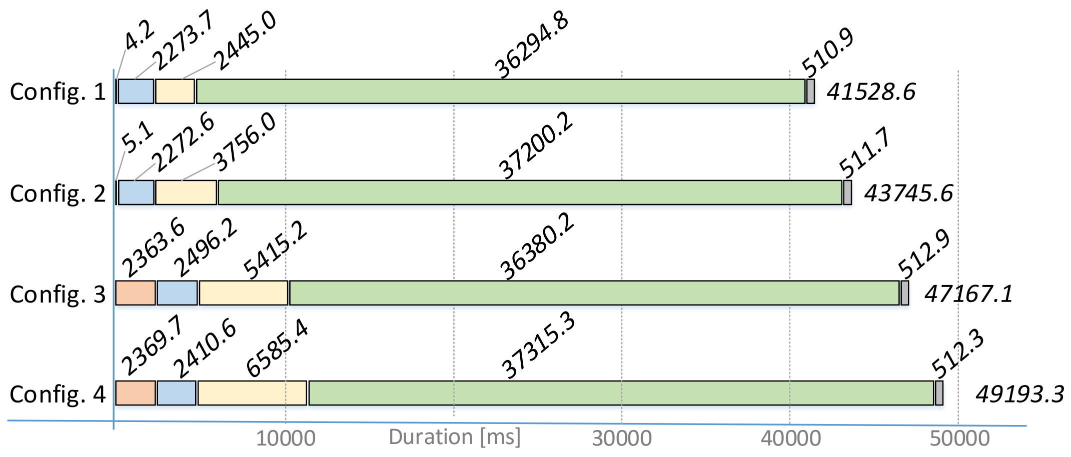


Figure 7.12: Impact of different security configurations on the update duration.

Table 7.2: Software update duration for different security configurations. The latency (in ms) of each step (except step 2) as well as the overall duration (also in ms) are presented.

Configuration	Step 1	Step 3	Step 4	Step 5	Step 6	Overall
1	4.2 (<0.1%)	2273.7 (5.5%)	2445.0 (5.9%)	36294.8 (87.4%)	510.9 (1.2%)	**41528.6**
2	5.1 (<0.1%)	2272.6 (5.2%)	3756.0 (8.6%)	37200.2 (85.0%)	511.7 (1.2%)	**43745.6**
3	2363.6 (5.0%)	2496.2 (5.3%)	5414.2 (11.5%)	36380.2 (77.1%)	512.9 (1.1%)	**47167.1**
4	2369.7 (4.8%)	2410.6 (4.9%)	6585.4 (13.4%)	37315.3 (75.9%)	512.3 (1.0%)	**49193.3**

this duration increase is quite significant, it must be accepted as all security mechanisms are required to guarantee a secure system execution.

Impact of different key lengths. The developed framework and its security concept allow different security configurations to tailor itself to the requirements of a local update environment (e.g., wireless software updates in a vehicle assembly line). Thereby, the length of the involved keys can be one important configuration parameter. In the following an evaluation of the impact of different key lengths of the RSA authentication key as well as the AES encryption key on the system performance is evaluated.

For both keys two different configurations parameters were used: AES key with either 128 or 256 bit; and RSA key with either 1024 or 2048 bit. As baseline the minimum security configuration RSA = 1024 bit and AES = 128 bit was chosen. For each security configuration ten sequential wireless software updates were performed using CESAR's main target ECU (i.e., AURIX) and thereby the software update duration was measured.

The collected results are presented in Figure 7.13 and show that varying the key length of the RSA-based authentication has a significant stronger impact on the overall update

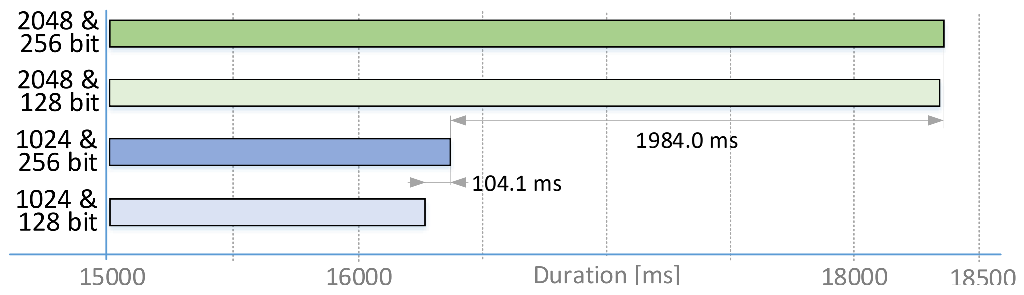


Figure 7.13: Impact of different key lengths on the wireless software update duration.

Table 7.3: Performance evaluation. TPM chip compared to the JBC software library and performance of the final, hybrid solution.

	Min [ms]	Max [ms]	Average [ms]
TPM	6328.0	6588.0	6415.5
JBC	575.0	701.0	635.8
Hybrid	3525.0	3690.0	3577.9

duration compared to different AES encryption key lengths. However, this is only relevant if a session is only established to perform one software update. In case that a session is established to perform several consecutive updates and/or to additionally run wireless diagnostics such as in a typical service center scenario, the authentication step only has to be performed once.

TPM vs. JBC performance. A secure key storage is an essential pillar of most comprehensive security solutions. Often TPMs are utilized to store secret keys and other sensitive data. Besides storing these keys and data, TPMs also offer different cryptographic functions such as the creation as well as verification of digital signatures.

As discussed in Section 6.2.4 the TPM as well as the JBC can be used to perform the RSA-based authentication. In the final concept a hybrid solution, where the TPM as well as the JBC library are involved in the authentication step, was developed as i) the TPM is quite slow compared to the JBC library and ii) JBC cannot provide the ability of securely store keys in dedicated memory.

In the following the performance of the JBC library and the utilized TPM is compared w.r.t. the cryptographic functions used in the authentication step. Therefore, the authentication step between a WVI and a DT is performed 20 times first using the TPM, second utilizing JBC, and third employing the hybrid solutions, and thereby the duration of these operations is measured using CESAR.

The gathered results, as presented in Table 7.3, clearly show that the JBC library outperforms the TPM chip. However, the most important advantage of the TPM is its ability to securely store sensitive data and keys, and thus the final concept uses a hybrid solution where the fast JBC library is combined with the secure storage of the TPM. The performance difference between JBC and the TPM, a HW chip dedicated to perform cryptographic operations, is analyzed in more detail in [37].

Chapter 8

Conclusions and Future Work

In this chapter the contributions of this thesis are summarized, limitations of the developed wireless software update solution are discussed, and potential for future work and research is outlined.

8.1 Contributions

A main goal of this thesis is to elaborate an end-to-end solution for locally-performed wireless automotive software updates. The proposed solution therefore consists of an architecture allowing secure software distribution realized using Blockchain technology, as proposed in Section 4.4, and EASE-UP, the framework for efficient and secure wireless automotive LSUs.

EASE-UP provides a basic wireless software update protocol that already significantly increases the flexibility of the update process, as it replaces the wired point-to-point links between the DT and a vehicle, supports the use of handheld devices, and allows to work on several vehicles at the same time (e.g., vehicle 1 receives a software update while vehicle 2 is diagnosed). In addition, EASE-UP provides advanced software update mechanisms, namely parallel (see Section 5.3.1) and partial (see Section 5.3.2) software updates. The performed experiments in the developed testbed infrastructure clearly show the benefit of employing such advanced update mechanisms, as they allow to significantly speed up the update process and thus to increase the efficiency of EASE-UP.

Besides efficiency, also security is essential for LSUs and thus a comprehensive security concept (see Section 6.2) is developed to protect EASE-UP against a wide range of external as well as internal attacks. One important aspect in the design of this security concept is to ensure that the security solution is suitable for different LSU scenarios. A structured system design approach based on measurable security proposed in this thesis (see Section 6.1) allows to analyze a system w.r.t. security, resulting in the most suitable security configuration for all considered scenarios. This approach is used to analyze EASE-UP and employed to identify security requirements w.r.t. LSUs. As a result, the designed security concept allows different configurations and is adjustable to the needs of the targeted LSU scenarios.

The developed software update mechanisms as well as the employed security features potentially have a significant impact on the efficiency of a wireless software update system.

To evaluate the actual impact of the developed update mechanisms and the identified security configurations on the duration of wireless software updates, a comprehensive testbed infrastructure is proposed in Section 7.1. Various experiments and tests described in Section 7.2 analyze different aspects of the developed solution. The gathered results thereby underpin the benefits of employing EASE-UP's advanced software update mechanisms.

8.2 Limitations and Future Work

In this section limitations of the developed system are described and the potential for future work and research is discussed.

Secure software distribution.

The secure software distribution is very likely the aspect of this thesis with the most potential for future work and research. Although the proposed architecture is fully defined as well as a proof-of-concept implementation already exist, one can further improve the distribution process itself. This could potentially be done by the introduction of caching mechanisms on the CHs of the overlay network. Such a mechanism could significantly speed up the distribution process for software required by lots of entities within a specific area (e.g., an update specific to European vehicles due to new regulations will require software updates for lots of vehicles within Europe). Furthermore, we plan to investigate how to use the Blockchain system to gathered diagnostic data from the vehicles. Thus, the direction of the data flow is reversed, as information about a vehicle is now sent back to the OEM and/or the supplier companies.

This idea can be further generalized by elaborating a generic automotive data exchange platform based on Blockchains. Such a platform would allow to provide different entities such as city planners, traffic management systems, insurance companies, as well as vehicle manufactures and suppliers with valuable data collected by smart connected vehicles while giving the owner of the vehicle full control over their data. The vehicle owner can decide which entities are allowed to access her/his data and, furthermore, if the data will be enriched with user-specific information (e.g., who is driving) or if the data is only shared in a anonymous manner. Additionally, such a generic data sharing platform will allow to realize different business models (i.e., what are the benefits for the vehicle owner when sharing her/his data) and help to build a new ecosystem for connected vehicles.

Bring EASE-UP to life.

EASE-UP is quite advanced and was already extensively tested within the developed testbed infrastructure. However, it is still a long way to go until EASE-UP can be used in a real-world automotive application. One important step would be to further develop the current prototypes with a special focus on the plug-in WVI. Currently, our WVI prototype is based on a BeagleBone Black single-board computer equipped with an additional Printed Circuit Board (PCB) providing suitable interfaces to connect the WVI to the vehicle, to power the board using a battery, and to use the TPM required for the authentication process. To use the WVI in harsh environments such as a service center, the developed prototype (its software) needs to be ported onto an automotive computing platform compliant to the requirements of automotive applications (e.g., supporting the full temperature range).

Development of secure gateways.

EASE-UP supports both the use of a plug-in WVI but also a gateway device that is fully integrated within the communication system of a vehicle. Such a fully integrated WVI can be part of a smart gateway interconnecting the vehicle with the outside world. Smart gateways will offer different communication interfaces such as 3G/4G/5G, Wi-Fi, Bluetooth, and IEEE 802.11p and are currently heavily investigated by academia and industry.

We plan to intensify our research in the area of smart gateways and thereby especially focus on the requirements of wireless software updates and diagnostics. Additionally, as these smart gateways are worthwhile targets for various attacks, we will also work on security aspects related to such gateways.

End-to-end security also on ECU level.

EASE-UP is employing a comprehensive security concept to protect all involved entities, namely the WVIs, the DTs, and the handhelds, and supports the widely used Seed & Key mechanism to authorize a software update on ECU level. At the time of writing (2018), using such a Seed & Key mechanism for ECUs is state of practice in the automotive industry and different automotive diagnostic standards are supporting this mechanism. However, there is a movement towards using Secure Elements (SEs) on (critical) ECUs to protect the data exchange and support authentication/authorization steps.

The use of SEs on ECU level is an important steps w.r.t. highly connected vehicles and thus will also be in focus of our future research. Thereby, we will especially focus on how to integrate SEs used on ECU level into EASE-UP and particularly its security concept.

Chapter 9

Publications

This thesis is based on the following peer-reviewed journal articles, and conference papers (ordered by publication date):

A. M. Steger, M. Karner, J. Hillebrand, W. Rom, E. Armengaud, M. Hansson, and K. Römer. Applicability of IEEE 802.11s for Automotive Wireless Software Updates. In Proceedings of the 13th International Conference on Telecommunications (ConTEL). Graz, Austria. September 2015.

B. M. Steger, M. Karner, J. Hillebrand, W. Rom, and K. Römer. A Security Metric for Structured Security Analysis of Cyber-Physical Systems Supporting SAE J3061. In Proceedings of the 2nd International Workshop on Modelling, Analysis, and Control of Complex CPS (CPS Data). Vienna, Austria. June 2016.

C. M. Steger, C.A. Boano, M. Karner, J. Hillebrand, W. Rom, and K. Römer. SecUp: Secure and Efficient Wireless Software Updates for Vehicles. In Proceedings of the 19th Euromicro Conference on Digital System Design (DSD). Limassol, Cyprus. September 2016.

D. M. Steger, C.A. Boano, M. Karner, J. Hillebrand, W. Rom, and K. Römer. Generic Framework Enabling Secure and Efficient Automotive Wireless SW Updates. In Proceedings of the 21st International Conference on Emerging Technologies and Factory Automation (ETFA). Berlin, Germany. November 2016.

E. M. Steger, A. Dorri, S.S. Kanhere, K. Römer, R. Jurdak, and M. Karner. Secure Wireless Automotive Software Up-dates using Blockchains – A Proof of Concept. In Proceedings of the 21st International Forum on Advanced Microsystems for Automotive Applications (AMAA). Berlin, Germany. September 2017.

F. M. Steger, C.A. Boano, K. Römer, M. Karner, J. Hillebrand, and W. Rom. CESAR: a Testbed Infrastructure to Evaluate the Efficiency of Wireless Automotive Software Updates. In Proceedings of the 20th International Conference on Modeling, Analysis and Simulation of Wireless and Mobile Systems (MSWiM). Miami, USA. November 2017.

G. M. Steger, T. Niedermayr, C.A. Boano, K. Römer, M. Karner, J. Hillebrand, and W. Rom. An Efficient and Secure Automotive Wireless Software Update Framework. To appear in IEEE Transactions on Industrial Informatics (TII) in 2018.

Related book chapters not included in this thesis:

1. P. Azzoni, F. Rogo, C. Coveri, M. Steger, W. Rom, A. Fiaschetti, F. Liberati, and J. Noll. Applying SHIELD in New Domains. Book chapter in Measurable and Composable Security, Privacy, and Dependability for Cyberphysical Systems – The SHIELD Methodology, ISBN 978-1-138-04275-9, 2018.
2. A. Dorri, M. Steger, S.S. Kanhere, and R. Jurdak. Blockchain: A distributed solution to automotive security and privacy. IEEE Communications Magazine, pages 119–125, December 2017.

Paper A

M. Steger, M. Karner, J. Hillebrand, W. Rom, E. Armengaud, M. Hansson, and K. Römer. **Applicability of IEEE 802.11s for Automotive Wireless Software Updates.** *In Proceedings of the 13th International Conference on Telecommunications (ConTEL)*, pages 1–8. Graz, Austria. September 2015.

Summary. This paper introduces a basic architecture of an automotive wireless software update system based on the IEEE 802.11s mesh standard. It mainly focuses on the evaluation of the IEEE 802.11s-based vehicle interface and on the influence of the environment on this interface as well as on the wireless link (e.g., radio interference, shielding). Therefore, the paper first lists essential technical requirements of a wireless software update system (i.e., throughput, scalability, reliability, extendability, compatibility, security, and functional safety) and evaluate different wireless technologies w.r.t. these requirements. Thereby, the paper shows that IEEE 802.11s is the most suitable candidate for a wireless software update system. Second, the paper described different evaluation steps and the corresponding results. These results prove the applicability of IEEE 802.11s as wireless media for automotive software updates.

My contributions. I am the main author of this article and I carried out the different experiments showing the applicability of IEEE 802.11s for automotive applications and, in particular, for automotive software updates. I wrote the vast majority of the paper in collaboration and discussion with the co-authors, who significantly helped to identify the requirements w.r.t. the wireless media, to define the basic design of the software update system, and to evaluate the measurement results w.r.t. to the defined requirements.

©2015 IEEE.
ISBN: 978-1-4799-8972-0.
DOI: 10.1109/ConTEL.2015.7231190.

Link: http://ieeexplore.ieee.org/document/7231190/.

Applicability of IEEE 802.11s for Automotive Wireless Software Updates

Marco Steger[*], Michael Karner[*], Joachim Hillebrand[*], Werner Rom[*], Eric Armengaud[†], Martin Hansson[‡], Carlo Alberto Boano[§], and Kay Römer[§]

[*]Virtual Vehicle Research Center, Graz, Austria
[†]AVL LIST GmbH, Graz, Austria
[‡]Volvo Group Trucks Technology, Göteborg, Sweden
[§]Institute for Technical Informatics, Graz University of Technology, Graz, Austria
Email: marco.steger@v2c2.at

***Abstract*—Due to the rising number of electronic control units (ECU) in a vehicle and the growing complexity of the related software installed, a fast and efficient system for updating software is needed. Wireless software updates similar to firmware over the air updates for smartphones can be a suitable solution to solve this issue. In this paper we propose a wireless update system based on an IEEE 802.11s mesh network and describe related high-level requirements for such a system. Additionally the prototype of a wireless vehicle interface (WVI) is described. This interface is needed to maintain the wireless link as well as to forward the received data to the in-vehicle communication system and finally to the ECU. Existing diagnostic standards are applied to transfer and install the new software on the ECU.**

Furthermore, IEEE 802.11s-based network nodes and the WVI prototype are used to evaluate the applicability of IEEE 802.11s for a wireless update system used in the vehicle development phase. We performed indoor measurements as well as measurements inside two different vehicles to evaluate the influence of the shielding properties of a vehicle. The results of these measurements show that the used setup consisting of the WVI prototype and other IEEE 802.11s based nodes can be used to realize a wireless update system and is able to fulfil the defined system requirements.

I. Introduction

The number of electronic control units (ECU) in a vehicle is rising, and the used software (SW) is getting more and more complex, leading to a growing number of bugs in the automotive SW. Because of that, efficient SW updates for vehicular ECUs are getting more and more important. An emerging trend for consumer devices is to perform wireless SW updates. These so-called firmware-over-the-air (FOTA) updates can help to speed up the update process and reduce the related costs dramatically. All major mobile phone manufacturers are already using FOTA updates to provide new features and to fix bugs. This technology can help to reduce costs for customers as well as for OEMs, avoid product recalls, and increase consumer satisfaction.

In a vehicle, the size of current ECU SW can be about tens of megabytes for engine and transmission controllers, while infotainment systems (e.g., audio or navigation system) are usually the largest and most complicated software units, often exceeding 100MB [1]. Therefore, a fast and reliable wireless link to the vehicle is needed to ensure that the required data can be transferred very quickly.

To enable FOTA updates, the vehicle must be equipped with a wireless vehicle interface (WVI), which is used to transfer the received data (e.g., the new SW version) to the ECU concerned by using the in-vehicle communication system. The wireless link to the WVI can either be based on an external cellular network or use a local network infrastructure (e.g., based on Wi-Fi or Bluetooth).

For vehicle respectively ECU development a system based on a local wireless network seems to be more attractive and realistic. A development engineer will be able to use such a wireless update system to download new SW to a vehicle (via the WVI) from his office or to run wireless diagnostics.

In this paper we will introduce such a wireless update system based on the IEEE 802.11s mesh standard. Thereby we will mainly focus on the evaluation of the IEEE 802.11s-based vehicle interface and on the influence of the environment on the WVI and the wireless link (e.g., radio interference, shielding). The described wireless update system will mainly be used in the development phase of a vehicle respectively on an ECU, but in a more generalized form it will also work in workshops (vehicle maintenance) or even in the assembly line.

The rest of the paper is structured as follows. In Section 2 the system requirements are stated and different wireless technologies are discussed. Section 3 provides an overview of the related work on wireless SW updates and applications based on IEEE 802.11s. Section 4 describes the architecture of the wireless update system and the components involved. In Section 5 we describe the experimental setup and the performed measurements. The results of these measurements and an evaluation of the results can be found in Section 6. Section 7 provides a summary of the key contributions of this applied research and discusses future work.

II. System requirements and evaluation of wireless technology

In this section the requirements of the wireless update system are defined and a technology evaluation based on these requirements is performed.

A. Requirements on the wireless vehicle interface and the wireless update system

In the following the most important requirements on the WVI are stated and described. The requirements must be taken into account when the interface respectively the entire system is designed and implemented.

- Throughput: as described above the size of the SW installed on an ECU node is growing and can be between dozens and hundreds of megabytes. Therefore, the link requires high throughput to ensure a fast data transfer. In practice, diagnostic protocols like UDS [2] are used to handle the SW update on the in-vehicle communication system and confine the maximum response time of an ECU node. This time is also influenced by the end-to-end latency of the link (e.g., round-trip time for 'tester present' messages to keep programming session alive: wireless link + in-vehicle communication consisting of bus transmission times and the time the gateway(s) need to forward the data + the time the ECU needs to react on the request $\leq$ 2000ms).
- Parallel updates and scalability: the wireless update system shall be able to handle updates in parallel. This means that several vehicles can receive new SW at the same time. Therefore, one update (one SW version for a specific ECU) can be transferred to several vehicles (multi-cast, e.g., the same type of car/ECU) and the system shall be able to distribute the required data (different vehicle types, ECUs, and SW versions) via the wireless link in an efficient, fast way (e.g., using different channels per vehicle).
- Reliability: the wireless link may operate in harsh environments (e.g., concerning temperature, vibration) and will also be impaired by different sorts of radio interferences.
- Extendability: the distance to the vehicle, which shall receive new SW versions, will vary and may exceed the current transmission range of the wireless update system. In such a case there must be an easy way (with less or without any additional configuration) to extend the transmission range of the system (e.g., several hundred meters to cover the whole area of a company or a test track).
- Compatibility: laptops, smartphones, tablets, or dedicated hardware devices (in the context of automotive SW updates also called diagnostic tester devices) will be used to interact with the vehicle via the WVI and therefore the used technology shall be available for consumer electronics.
- Security: updating the SW of an ECU and adding a WVI to the existing vehicle architecture can be very critical and therefore the integrity of the transferred data as well as the integrity of the vehicle must be ensured. This paper focuses on the wireless system architecture as well as on the properties of IEEE 802.11s, not on the security layer on top of it. We will use the SHIELD methodology proposed in [3], [4] to continuously evaluate our system regarding security, privacy as well as dependability issues and to find the best configuration of the core components of our wireless update system. Additionally, security on higher network layers will be addressed as part of future research carried out within our project.
- Functional safety: for embedded automotive SW functional safety is an important issue and must also be considered when the SW of an ECU shall be updated. ISO 26262 [5] defines a safety life cycle by delineating different levels of safety requirements [6]. Regarding our wireless update system we will start from a higher level of functional abstraction (safety goal, functional safety requirements) to the more detailed levels of the technical realization of the system (technical safety requirements), down to the software (software safety requirements) and hardware levels (hardware safety requirements). For the implementation we will use knowledge about functional safety methods and work flows which we developed in several projects (e.g., SafeCer [6], VeTeSS [7], [8]) to identify all relevant functional safety requirements as well as continuously analyze and improve our system w.r.t. functional safety.
- Interconnection with other networks respectively (re)use of existing infrastructure: the interconnection to other networks (e.g., the network of a company) would be very beneficial for the wireless SW update scenario. The WVI or the diagnostic tester can connect to the network infrastructure of a company (via WLAN, Ethernet) to enable remote updates or to increase the range of the entire system.

B. Wireless technology evaluation

Based on the requirements stated above, a suitable wireless standard has to be chosen as the communication technology for our wireless update system. IEEE 802.15.4 is a very power-efficient protocol and is used in many different wireless sensor network (WSN) applications. However, this standard would be too slow for our purposes [9], [10] (raw bit rate of 250 Kbps, with a measured throughput $<$50Kbps). The throughput of Bluetooth Low Energy (BLE) is quite similar to IEEE 802.15.4 [11] (maximum application layer throughput is 236.7Kbps, typically $<$60Kbps), which also makes it unsuitable for our wireless update scenario.

A lot of research regarding vehicle-to-vehicle communication (IEEE 802.11p [12]) has been carried out in the last decades, but it is not clear when respectively if IEEE 802.11p will be integrated into vehicles. Currently, IEEE 802.11p platforms and interfaces are quite expensive compared to Bluetooth or IEEE 802.11 components. Laptops or hand-held devices such as smartphones or tablets probably won't be equipped with IEEE 802.11p hardware at all. So the availability of IEEE 802.11p components is currently insufficient. Additionally, the interconnection to other IEEE 802 networks is hard to realize because current automotive IEEE 802.11p stacks are not IP-based and a dual-stack solution would be needed to interconnect IEEE 802.11p and other infrastructural networks (e.g., a corporate network via Ethernet or Wi-Fi). Current IEEE 802.11 networks offer typically enough bandwidth to satisfy the needs of the described SW update scenario.

The IEEE 802.11b/g/n hardware is integrated in nearly every recent laptop and hand-held device, and a good fraction of this hardware can also be used for IEEE 802.11s (e.g., several Atheros and Qualcomm chips already support IEEE 802.11s).

IEEE 802.11 protocols are designed for large networks and therefore satisfy the scalability requirement. Extending the communication range of IEEE 802.11b/g/n network is possible but dedicated hardware is required. IEEE 802.11s networks can be extended easily by adding relay nodes (forwarding data to other nodes, multi-hop) without any configuration effort (in Section 3, an example regarding the range of the wireless update system is presented). Additionally, IEEE 802.11s was designed in a way that gateway nodes can be easily used to forward data to other IEEE 802 networks.

III. Related work

The use of FOTA updates in the automotive domain was already addressed in several works, which we summarize is this section. Additionally, some applications based on IEEE 802.11s are described to show the potential of IEEE 802.11s.

A. Wireless SW updates

A white paper from Redbend [1] summarizes the benefits of using FOTA updates in the automotive domain and shows the benefits of this technology for both OEMs and end users. In [13] the basic idea and the benefits of a dynamic SW update system are demonstrated. The system provides high-level abstractions for sensing and tuning automobile parameters. Using these abstractions, developers can achieve fuel efficiency, responsiveness, or safety goals and users can tune their vehicles at the granularity of individual trips, a capability we call personalized tuning. Several other scientific papers regarding SW updates over the air are available. These papers are mainly focusing on security and verification issues but the question of how the data can be transferred to the vehicle was neither addressed nor resolved.

A system where vehicles can be updated using an Internet connection to a portal server is proposed in [14]. The authors mainly focus on the secure data link between the vehicle and the portal and no in-depth information on how the data is transferred to the ECU or how a vehicle can connect to the network is provided. However, an overview on desired security properties for the network traffic of such an automotive system is given.

In [15], a security hardware module for vehicular ECUs, to handle the verification of new SW updates, is proposed. The module can also be used for data encryption respectively decryption, digital signatures, and authentication. The wireless interface and the wireless link are not described.

The authors of [16] focus on how transferred SW can be verified when it is flashed to the ROM. Therefore, an additional control system (as part of the ECU), which is responsible to handle the verification procedure, is defined. In [17] a classification of ECUs is stated and described. This approach is based on the assumption that different ECUs may require different levels of security. The idea is that an ECU, which was classified as very critical, is not allowed to be updated using FOTA updates.

All these articles focus on how wireless updates can be performed and on how the required data can be transferred to the vehicle in a secure way. However, no information about how the wireless link actually can be realized and which additional components and requirements are needed for such a system are given.

FOTA updates play a crucial role also in wireless sensor networks, as they allow to extend the software with additional features and to fix existing bugs. Over-the-air programming techniques for wireless sensor networks typically exploit their multi-hop communications to transfer the new software to all nodes in the network and need to meet also the limited power, processing, and storage capabilities of sensor nodes [18]. The same principles can be used in vehicular settings by using multi-hop capabilities to carry out a fast and efficient firmware update on several vehicles situated in a large area.

B. IEEE 802.11s applications and performance analysis

We will use IEEE 802.11s as communication media for our wireless update infrastructure. Therefore we carried out a literature research to get a better overview on the performance and on the possibilities, as well as on the limitations of IEEE 802.11s. In the following, we present the results of our literature research.

In [19] an overview on IEEE 802.11s is given. The paper starts with a brief explanation of IEEE 802.11 networks and then focuses on 11s mesh networks. Frame structure, channel selection, power management, security, and path selection are described. Additionally, measurements in a test-bed consisting of ten nodes (basic performance measurements, enforced multi-hop) were performed.

In [20] and [21], IEEE 802.11s is used as a backbone network for V2X[1] networks. The main idea is to replace the wired connections between the RSUs (road side units) and the V2X servers by wireless ones. The authors of both papers use simulation to verify the system performance.

An evaluation of the impact of uncontrolled traffic sources on real-time communication in IEEE 802.11s networks is described in [22]. The results are based on simulations using the *ns-3* simulator. The idea is to set up a WSN with real-time constraints and simulate the impact of radio interference (simulated HTTP traffic).

Other papers focus on improving the standard Hybrid Wireless Mesh Protocol (HWMP), the routing protocol used in IEEE 802.11s mesh networks, to increase the fairness in mesh networks and to make IEEE 802.11s networks more energy efficient. In [23] an energy efficient HWMP scheme (eHWMP) is proposed, which shall help to increase the overall network lifetime. Simulations are carried out to show the positive impact of eHWMP on network lifetime. The authors of [24] propose an energy-optimization-based path selection algorithm which can be incorporated in the standard HWMP. Simulations are carried out to evaluate the performance of the path selection algorithm. The authors of [25] propose a power efficient mesh application with nearly identical throughput as

[1]General term for vehicle to vehicle and vehicle to infrastructure communication.

normal 11s. The performance of the system was evaluated based on simulation results.

In [26] the implementation of IEEE 802.11s nodes based on open80211s[2] is explained and test-bed measurements (throughput vs. hop-count) were carried out. The authors also address the gateway functionality of 802.11s to other (wireless) networks. In [28] a campus-wide mesh network was used to test new mesh application. A proprietary mesh network (no IEEE 802.11s) based on madWIFI[3] was defined. The results of the experiments (a very basic evaluation of a video streaming application) are also briefly described.

A big portion of the mentioned articles use network simulators to validate the described applications. In the automotive domain, the environment can be quite rough for a wireless network architecture (e.g., regarding vibrations, temperature, and radio interference). Therefore, in this work, we perform real-world measurements instead of simulations to evaluate the performance and the applicability of an IEEE 802.11s based wireless update system in a vehicular setting.

IV. System description

The wireless update system shown in Figure 1 is based on a IEEE 802.11s mesh network infrastructure, where several vehicles, hand-held devices such as smartphones and tablet and diagnostic tester devices (the data source, where the new ECU SW version is stored) like laptops, PCs or dedicated hardware devices (mainly used in workshops) are connected to each other either directly or via another device/node. Because of the mesh characteristic of the IEEE 802.11s standard, additional devices can be used as relay nodes (e.g., a parked vehicle or a placed relay node) between two end nodes. If there is no direct connection between the source (e.g., the diagnostic tester) and the target (e.g., the WVI), a relay node can be placed in between without any extra configuration of the network and the nodes. This also means that the transmission range of the wireless system can be extended easily by (temporarily) adding/placing relay nodes. If a development engineer wants to download SW to an ECU of a vehicle parked outside he can either directly connect to the WVI (if in transmission range) or place a relay node (e.g., at a window near the parking position of the vehicle) and use this node to extend the range to ensure the connection with the WVI.

The hand-held devices can be used to schedule, trigger, and monitor the update process and to run wireless diagnostics. The diagnostic tester can be seen as the data source. The data (SW binary) can be located directly on the device or the device can be connected to a backbone network (e.g., OEM SW server) via an Internet link. The SW binary is sent to the WVI via the IEEE 802.11s link using TCP or UDP. In the next step the WVI will forward the data to the ECU, which shall be updated using the in-vehicle communication system.

Figure 1 shows a higly simplified model of the in-vehicle communication system. In reality, a vehicular communication system consists of dozens of ECUs, several different bus systems (e.g., CAN, FlexRay, LIN) and a central gateway device,

[2]open80211s is an open-source implementation of the recently ratified IEEE 802.11s wireless mesh standard [27].

[3]http://madwifi-project.org/

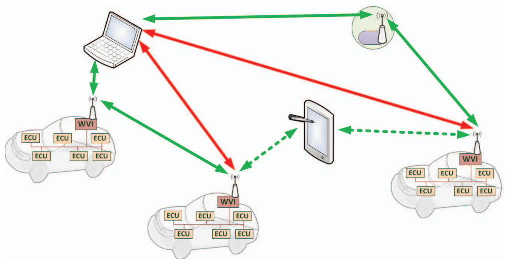


Fig. 1. The top-level communication model of the wireless update system.

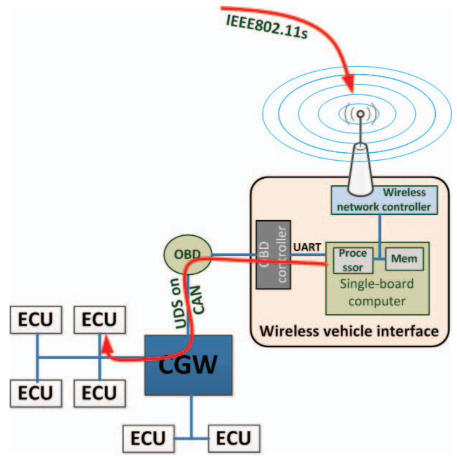


Fig. 2. Simplified block diagram of the WVI and the links to the in-vehicle communication system and the IEEE 802.11s mesh network.

which is used to interconnect all bus systems and the ECUs. The in-vehicle communication system can be accessed from outside via the OBD interface. The interface is mainly used for vehicle diagnostics (e.g., reading error codes) but can also be used to transfer data (in our case the new SW version) to an ECU inside the vehicle. Therefore diagnostic standards like Unified Diagnostic Services (UDS) or Universal Measurement and Calibration Protocol (XCP), which are running on top of buses such as CAN or automotive Ethernet, can be used.

It is important that such a diagnostic standard is used to guarantee the backward compatibility and the system acceptance inside the automotive domain of our solution.

The first prototype of our WVI consist of a Beaglebone black (BBB), a single-board computer, a IEEE 802.11(s) Wi-Fi stick (TL-WN722N) and an OBD-Controller and is implemented according to the system requirements stated in Section 2. In Figure 2 the block diagram of the WVI prototype is presented. Additionally, the connection to the in-vehicle communication system (via the OBD interface) as well as to the ECU node is shown.

V. Experimental setup

In the considered scenario the test vehicle is parked in front of the building where the office of the development engineer is located. To test a new SW version the engineer wants to download new SW to the vehicle from the office.

The prototype of the WVI described above was used to perform some measurements to evaluate the feasibility of our system setup. The basic idea was to get a feeling about the range of the TL-WN722N sticks using IEEE 802.11s in and around a vehicle. Additionally, the stick was used to perform a spectral scan (in the 2.4GHz band) in two different cars.

In the following the measurement setup is described in more detail. The results of the measurements are stated and discussed in the next section.

A. Indoor Measurements

In the first step we used two nodes to measure the transmission range of the system inside a building. Therefore we performed indoor measurements in the Virtual Vehicle office building. The first node was used as static node (NodeB) and second node (NodeA) was moved around in the building (on second floor, both nodes on the same floor for the whole measurement, a mix of thin wooden and concrete walls). Thereby we evaluated the impact of the distance between the nodes on the link quality. Among others, we measured the signal strength and used iperf[4] to evaluate the link quality. For the measurement, the Debian default rate control algorithm (Minstrel[5]) was enabled, but the rate remain at 54Mbps all the time. The multi-hop ability of 11s based network was not used in this measurement setup. This means that no relay nodes were used to increase the transmission range.

B. The impact of a vehicular environment on the IEEE 802.11s link

The WVI prototype was placed inside a vehicle (BMW X3) and the link properties to other nodes located inside as well as outside the vehicle was measured.

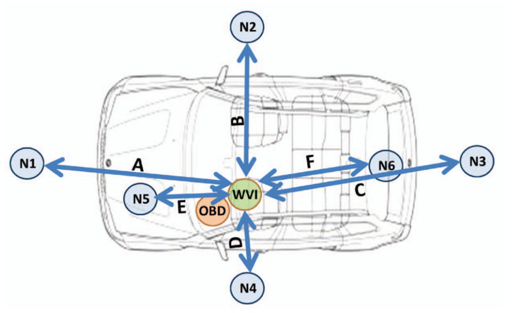


Fig. 3. Position of the used 11s node inside and around the BMW X3 and the measured links.

[4]https://iperf.fr/

[5]See https://wireless.wiki.kernel.org/en/developers/documentation/mac80211/ratecontrol/minstrel for more information

TABLE I. The indoor link properties between a two nodes

Distance [m]	Signal$_{NodeA}$ [dBm]	Signal$_{NodeB}$ [dBm]	Metric
0	-16	-17	152
10	-42	-44	152
20	-46	-51	152
30	-66	-66	152
40	-59	-64	152
50	-68	-73	152
60	-64	-65	152
70	-69	-73	152

In Figure 3 the location of the WVI (near the OBD interface) and the other nodes is shown. The WVI as well as the other nodes consists of a BBB and a plugged in TL-WN722N stick. The distance between the vehicle and the nodes outside the vehicle (N1-N4) were about 5m and the data rate was 54Mbps (fixed rate with disabled rate control) for all performed measurements.

C. Spectral scan of the 2.4 GHz band performed in two different vehicles

The actuators in a vehicle will create different kinds of interference. These interferences may also influence the link quality between the WVI and other nodes in the transmission range of the wireless update system. To get a better feeling of how this interference can look like, we performed some measurements in a conventional diesel car (BMW X3) and also in a electric car (Citroen C-Zero). In practice, additional interferences because of other wireless networks (e.g., WLAN and Bluetooth) from outside but also inside the vehicle (e.g., electronic devices of the passengers or the infotainment system) can occur and may also influence the link quality. However, in this paper we focus on the evaluation of the influence of the vehicle itself (infotainment off). Therefore, we first measured a baseline (no vehicle-based interferences, ignition and engine off) to check that no other sources of interference will influence our spectral scans.

The hardware used to collect the data consisted of a BBB and the TL-WN722N stick (especially the ability of the built-in Atheros ATH9K_HTC chip to perform a spectral scan in the 2.4GHz band). An adapted version of the open source spectrum analyzer tool FFT_eval[6] was used to create the plots shown in the following section. The resulting plots are mainly influenced by 1) the signal strength and 2) the density over several measurements. Hence, the blurry look of the plots is an additional level of information.

VI. Measurements and Evaluation

In this section the measurement results are described and evaluated.

A. Indoor measurements

The first measurements with 802.11s-based nodes was done indoors in the Virtual Vehicle office building. In Table I the link properties as a function of the distances between the nodes is shown.

Along with the hop count (number of hops respectively other nodes between the sender and the target node) the link

[6]https://github.com/simonwunderlich/FFT_eval

metric is used to find the best path through a mesh network between two nodes. In IEEE 802.11s networks the metric is a combination of the frame error rate and the bit rate. In our measurements, the metric remains constant because the frame error rate (error rate at the MAC layer, retries on HW are not counted) stays 0 during the measurement and the metric remains unchanged. In Figure 4 the plots of the received signal strength are shown.

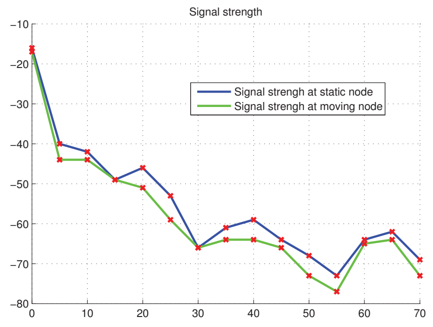


Fig. 4. The indoor link between a static and a moving node.

The measured curves are quite similar to access-point-based Wi-Fi like IEEE 802.11b/g/n and the indoor transmission performance (0 transmission errors at 70m distance and 54Mbps) is sufficient for the described scenario.

B. *IEEE 802.11s based nodes inside and around a BMW X3*

In Table II the results of the measurement are stated. For each link marked in Figure 3 the signal strength and the link metric is stated.

TABLE II. The IEEE 802.11s link properties between the WVI and a second node

Link	Distance [m]	Signal$_{WVI}$ [dBm]	Signal$_{Node}$ [dBm]	Metric$_{WVI}$	Metric$_{Node}$
A	7	-61	-63	152	152
B	6	-59	-62	152	152
C	7.5	-58	-69	152	152
D	5.5	-55	-59	152	630
E	1.5	-46	-49	152	152
F	2	-45	-47	152	152

Except for link D the metric still remains constant because no frame errors occurred during the measurement. For link D one frame error occur during our measurements (frame sent to the WVI) and because of that the metric is higher. We also did this measurement with reduced data rate (1Mbps) and got a constant metric value (8193) for all links. So the metric value stays constant but the value is way higher because of the reduced data rate (1Mbps compared to 54Mbps). The shielding properties of the vehicle significantly influence the received signal strength. The distance between the nodes outside the vehicle and the WVI node is just about 5m but the measured signal strength is between -45 and -69dBm. Compared to the results of the indoor measurements this range equals a node distance between 20 and 50m. As presented in Table II the emitted signal strength measured at two nodes located inside a vehicle (links E and F) is quite low compared to the other results (links A-D).

C. *Spectral scan inside a BMW X3*

In the previous section the shielding properties of a vehicle were measured and explained. The results show that the shielding due to the metal, glass, etc. significantly reduces the signal strength at the receiving node. Additionally, the actuators in a vehicle will create all kinds of interference which will also influence the robustness of wireless update systems operating in the 2.4Ghz band. The following spectral scans shall help to understand how this interference can look like. In Figure 5 the spectral scan of the BMW X3 (Diesel) during the start process (ignition off to engine started) is shown. Some significant peaks can be found in the spectrum. The results of several scans show that these peaks can appear in all channels of the 2.4GHz band.

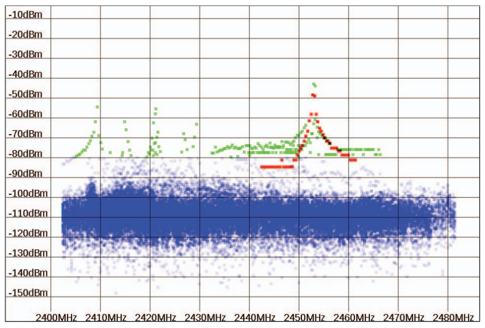


Fig. 5. Starting the engine of our BMW X3 (blue: samples below -80dBm, green: -80dBm and above, red: one specific sample).

The same kind of peaks can also be found in the scans shown in Figures 6 and 7, where scans of the same BMW with running engine are presented. These scans clearly reveal that interference can appear in all channels of the 2.4GHz frequency band. Although, temporary there are channels without significant interferences (see, e.g., Figure 7: no significant interference between 2460 and 2480MHz).

D. *Spectral scan inside a Citroën C-Zero (e-car)*

In an electric car quite different actuators may be in use and therefore the occurring interference may also differ. Because of that we decided to create the same kind of spectral scans also for an electric car. In Figure 8 the spectral scan of Citroën C-Zero in parking mode (ignition on but vehicle in standstill) is shown. In the collected frequency spectrum similar peaks can be found but there are more peaks and more interference in general distributed over the whole spectrum.

In the next step we collected spectral data while driving around with the e-car (constant speed, approximately 30km/h) to find out if the interferences are different than in standstill. The results of these measurements are shown in Figures 9 and 10.

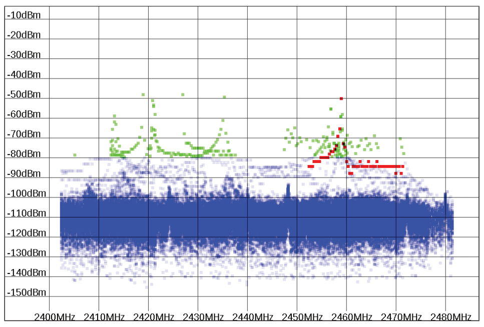


Fig. 6. First spectral scan of the BMW X3 with running engine (blue: samples below -80dBm, green: -80dBm and above, red: one specific sample).

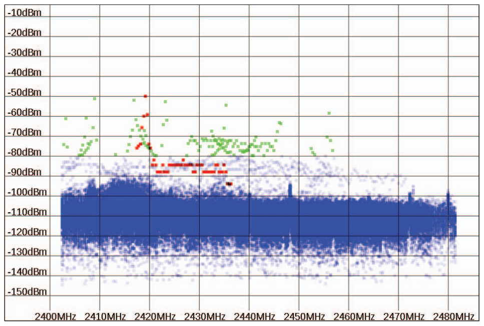


Fig. 7. Second spectral (approximately 1 minute after the first scan) scan of the BMW X3 with running engine (blue: samples below -80dBm, green: -80dBm and above, red: one specific sample).

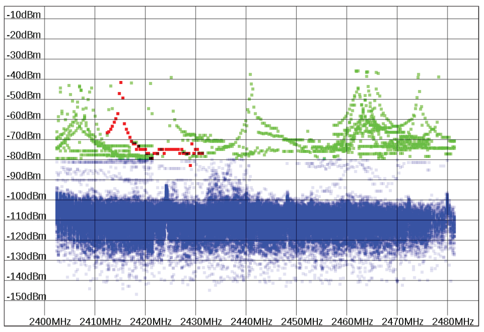


Fig. 8. Citroën with ignition on but in standstill (blue: samples below -80dBm, green: -80dBm and above, red: one specific sample).

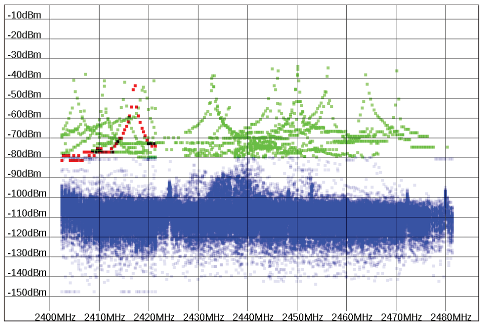


Fig. 9. First scan of Citroën e-car at 30km/h (blue: samples below -80dBm, green: -80dBm and above, red: one specific sample).

The scans show that there is no significant difference between vehicle in standstill and the vehicle driving at constant speed. Compared to the scans of the BMX X3 the frequency spectrum of the e-car contains more samples above the -80dBm limit. Although there are also some regions with less interference (see 2430-2440MHz in Figure 9 and 10).

VII. Conclusions and Future Work

In this paper we described the design and the high-level requirements for an IEEE 802.11s based wireless update system for automotive software. Additionally, a prototype of a wireless vehicle interface (WVI) was presented. Different measurements and an evaluation of an IEEE 802.11s based network in an automotive context was performed, and the result were presented in the previous sections.

Our experimental results clearly show that IEEE 802.11s can be used as the basis technology for a wireless SW update system. Indoor measurements prove that a robust and reliable link between two nodes in a distance of up to 70m can be realized. Shielding properties of vehicles significantly influence the transmission range and the signal strength. However, a reliable IEEE 802.11s based link between the WVI inside the vehicle and a diagnostic device (e.g., a laptop, tablet or smartphone) or a relay node (nodes outside the vehicle) can be achieved. Based on our experiments, we were able to show that an IEEE 802.11s-based wireless update system can be realized with the components used (mainly the WVI consisting a single-board computer and a Wi-Fi stick). Additionally, we also measured and analyzed the frequency spectrum of a conventional car (BMW X3) and an e-car (Citroën C-Zero). The results show that there is interference in the 2.4GHz band which must be taken into account. Especially for nodes with smaller antennas (e.g., a PCB antenna instead of the rod antenna of the TL-WN722N) these interferences together with the shielding properties of a vehicle can be critical.

In the next step we plan to test such a node with a smaller (PCB) antenna and to evaluate if it still can be used for our purpose. This smaller solution can then be used as plug-in device for wireless updates and diagnostics. Additionally, we will also think about an integrated solution, where the WVI is part of the in-vehicle communication system, thereby being able to use the antenna(s) of the vehicle. The antenna diversity will help to improve the link quality and to avoid attenuation issues. In the next months we will focus on the advancement of the WVI including the implementation of the gateway functionality to forward data received from the wireless update system to the ECU node. A security layer will also be added

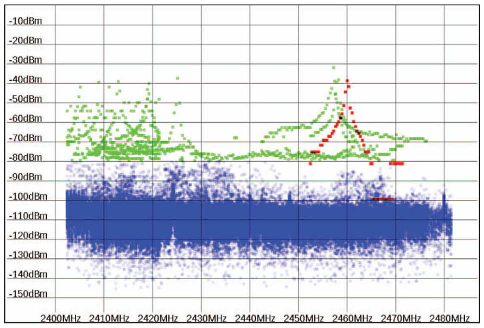


Fig. 10. Second scan (approximately 1 minute after the first scan) of Citroën e-car at 30km/h (blue: samples below -80dBm, green: -80dBm and above, red: one specific sample).

to ensure the integrity of the transferred data as well as the integrity of the vehicle.

Acknowledgments

The research from DEWI project (www.dewi-project.eu) leading to these results has received funding from the ARTEMIS Joint Undertaking under grant agreement n° 621353. The authors acknowledge the financial support of the COMET K2 - Competence Centres for Excellent Technologies Programme of the Austrian Federal Ministry for Transport, Innovation and Technology (BMVIT), the Austrian Federal Ministry of Science, Research and Economy (BMWFW), the Austrian Research Promotion Agency (FFG), the Province of Styria and the Styrian Business Promotion Agency (SFG).

References

[1] Redbend Software, "Updating Car ECUs Over-The-Air (FOTA)," *White Paper*, p. 14, 2011.

[2] ISO, "Road vehicles Unified diagnostic services (UDS) Specification and requirements," ISO 2006, Tech. Rep., 2006.

[3] J. Noll, I. Garitano, S. Fayyad, E. Åsberg, and H. Abie, "Measurable security, privacy and dependability in smart grids," *Journal of Cyber Security*, vol. 3, pp. 371–398, Apr. 2015.

[4] I. Garitano, S. Fayyad, and J. Noll, "Multi-metrics approach for security, privacy and dependability in embedded systems," *Wireless Personal Communications*, vol. 81, no. 4, pp. 1359–1376, 2015.

[5] ISO, "Road vehicles – Functional safety – Part 1: Vocabulary," ISO, Tech. Rep., 2011.

[6] H. Martin, M. Krammer, B. Winkler, and C. Schwarzl, "Model-based engineering workflow for automotive safety concepts," *SAE 2015 World Congress and Exhibition*, 2015.

[7] M. Karner, M. Krammer, and A. Fuchs, "System level modeling, simulation and verification workflow for safety-critical automotive embedded systems," *SAE 2015 World Congress and Exhibition*, 2014.

[8] M. Krammer, P. Stirgwolt, and H. Martin, "From natural language to semi-formal notation requirements for automotive safety," *SAE 2015 World Congress and Exhibition*, 2015.

[9] M. Petrova, J. Riihijarvi, P. Mahonen, and S. Labella, "Performance study of IEEE 802.15.4 using measurements and simulations," *IEEE Wireless Communications and Networking Conference, 2006. WCNC 2006.*, vol. 1, no. c, pp. 487–492, 2006.

[10] F. Mieyeville, W. Du, I. Daikh, and D. Navarro, "Wireless Sensor Networks for active control noise reduction in automotive domain," *2011 The 14th International Symposium on Wireless Personal Multimedia Communications (WPMC)*, pp. 1–5, 2011.

[11] C. Gomez, J. Oller, and J. Paradells, "Overview and evaluation of bluetooth low energy: An emerging low-power wireless technology," *Sensors (Switzerland)*, vol. 12, no. 9, pp. 11 734–11 753, 2012.

[12] I. S. for Information technology, "Local and metropolitan area networks-Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY):Wireless Access in Vehicular Environments," IEEE, Tech. Rep., 2010.

[13] T. Flach, N. Mishra, L. Pedrosa, C. Riesz, and R. Govindan, "CarMA: Towards Personalized Automotive Tuning," University of Southern California, Tech. Rep. 11-921, 2011.

[14] D. K. Nilsson and U. E. Larson, "Secure firmware updates over the air in intelligent vehicles," *IEEE International Conference on Communications*, pp. 380–384, 2008.

[15] M. S. Idrees, H. Schweppe, Y. Roudier, M. Wolf, D. Scheuermann, and O. Henniger, "Secure automotive on-board protocols: A case of over-the-air firmware updates," *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 6596 LNCS, pp. 224–238, 2011.

[16] D. Nilsson, L. S. L. Sun, and T. Nakajima, "A Framework for Self-Verification of Firmware Updates over the Air in Vehicle ECUs," *2008 IEEE Globecom Workshops*, pp. 1–5, 2008.

[17] D. Nilsson, P. Phung, and U. E. Larson, "Vehicle ECU classification based on safety-security characteristics," *Road Transport Information and Control - RTIC 2008 and ITS United Kingdom Members' Conference, IET*, pp. 1–7, 2008.

[18] S. Brown and C. J. Sreenan, "Software updating in wireless sensor networks: A survey and lacunae," *Journal of Sensor and Actuator Networks*, vol. 2, no. 4, pp. 717–760, 2013.

[19] G. R. Hiertz, D. Denteneer, S. Max, R. Taori, J. Cardona, L. Berlemann, and B. Walke, "IEEE 802.11s: The WLAN mesh standard," *IEEE Wireless Communications*, pp. 154–160, 2010.

[20] D. T. Tuan, S. Sakata, and N. Komuro, "Priority and admission control for assuring quality of I2V emergency services in VANETs integrated with Wireless LAN Mesh Networks," *ICCE 2012*, pp. 91–96, 2012.

[21] S. Chakraborty and S. Nandi, "IEEE 802.11s mesh backbone for vehicular communication: Fairness and throughput," *IEEE Transactions on Vehicular Technology*, vol. 62, no. 5, pp. 2193–2203, 2013.

[22] C. M. D. Viegas, F. Vasques, and P. Portugal, "Evaluating the Impact of Uncontrolled Traffic Sources upon Real-Time Communication in IEEE 802 . 11s Mesh Networks," *Industrial Informatics (INDIN), 2014 12th IEEE International Conference*, p. 4, 2014.

[23] M. A. Ng and K.-l. A. Yau, "An Energy-Efficient Hybrid Wireless Mesh Protocol (HWMP) for IEEE 802.11s Mesh Networks," pp. 17–21, 2013.

[24] M. M. Mhlanga and T. O. Olwal, "Energy Optimization based Path Selection Algorithm for IEEE 802 . 11s Wireless Mesh Networks," no. September, pp. 13–15, 2011.

[25] S. Chen and G. M. Muntean, "E-Mesh: An energy-efficient cross-layer solution for video delivery in wireless mesh networks," *IEEE International Symposium on Broadband Multimedia Systems and Broadcasting, BMSB*, 2012.

[26] P. Pandey, S. Satish, J. Kuri, and H. Dagale, "Design & implementation of IEEE 802.11s mesh nodes with enhanced features," *2009 IEEE 6th International Conference on Mobile Adhoc and Sensor Systems, MASS '09*, pp. 639–644, 2009.

[27] "open80211s – an open-source implementation of the recently ratified ieee 802.11s wireless mesh standard," http://open80211s.org/open80211s/.

[28] M. Cesana, L. Fratta, M. Gerla, E. Giordano, and G. Pau, "C-VET the UCLA campus vehicular testbed: Integration of vanet and mesh networks," *2010 European Wireless Conference, EW 2010*, pp. 689–695, 2010.

Paper B

M. Steger, M. Karner, J. Hillebrand, W. Rom, and K. Römer. **A Security Metric for Structured Security Analysis of Cyber-Physical Systems Supporting SAE J3061.** *In Proceedings of the 2nd International Workshop on Modelling, Analysis, and Control of Complex CPS (CPS Data)*, pages 1–6. Vienna, Austria. June 2016.

Summary. In this paper the DEWI security metrics, a structured and detailed system analysis with respect to security, are proposed. First, the new SAE J3061 standard (Cyber-security Guidebook for Cyber-Physical Vehicle Systems) [52] is described and some insight on how it can be used to evaluate and analyze a wireless software update system given. Thereby, the paper describes the development process described in this standard, lists all essential steps of the concept phase (i.e., an important part of the described development process), and highlights how the DEWI security metrics can support these steps. Thereafter, the DEWI security metrics are explained in more detail and the paper shows that the DEWI security metrics can be utilized to perform a structured security analysis with comparable and reusable results in different domains. Finally, a case study is presented showing how the DEWI security metrics (in combination with the new SAE security standard) can be applied to a real use case.

My contributions. As main author of this paper I evaluated the SHIELD multi-metrics and identified its limitations, defined (with significant support by the co-authors) the DEWI security metrics to overcome these limitations, described the benefits of utilizing the DEWI security metrics to support SAE J3061, and carried out a case study (security analysis of a wireless vehicle interface) to prove the applicability of the DEWI security metrics. The co-authors eminently supported this work by providing detailed reviews and fruitful discussions about the DEWI security metrics and the connection the the SAE standard.

©2016 IEEE.
ISBN: 978-1-5090-1154-4.
DOI: 10.1109/CPSData.2016.7496425.

Link: http://ieeexplore.ieee.org/document/7496425/.

A Security Metric for Structured Security Analysis of Cyber-Physical Systems Supporting SAE J3061

Marco Steger*, Michael Karner*, Joachim Hillebrand*, Werner Rom*, and Kay Römer†

*Virtual Vehicle Research Center, Graz, Austria
†Institute for Technical Informatics, Graz University of Technology, Graz, Austria
Email: marco.steger@v2c2.at

***Abstract*—Complexity in modern vehicles has increased dramatically during the last years due to new features and applications. Modern vehicles are connected to the Internet as well as to other vehicles in close proximity and to the environment for different novel comfort services and safety-related applications. Enabling such services and applications requires wireless interfaces to the vehicle and therefore leads to open interfaces to the outside world. Attackers can use those interfaces to impair the privacy of the vehicle owner or to take control (of parts of) the vehicle, which strongly endangers the safety of the passengers as well as other road users. To avoid such attacks and to ensure the safety of modern vehicles, sophisticated structured processes and methods are needed.**

In this paper we propose a security metric to analyse cyber-physical systems (CPS) in a structured way. Its application leads to a secure system configuration with comparable as well as reusable results. Additionally, the security metric can be used to support the conceptual phase for the development of CPS specified in the new SAE security standard SAE J3061. A case study has been carried out to illustrate the application of the security metric.

I. Introduction

In the near future vehicles will be part of the Internet of things and different cloud services already known from smartphones will also be available. Additionally, other applications like remote diagnostics and remote software updates (e.g., enabling new features or increasing engine power) will be possible. All these services require a wireless link between the wireless interface(s) of the vehicle and the OEM network or even the Internet. To this end either a direct 3G/4G/5G connection or a short-range wireless technology like WiFi (as last mile network) can be used between the vehicle and the Internet. Adding a wireless interface to a modern vehicle will enable a big variety of different new services and applications but also leads to an open and attackable interface to the outside world. Thus, a wireless vehicle interface (WVI) is a critical component with respect to privacy, security and, in further consequence, vehicle safety. Today already different attacks using the infotainment system of a vehicle are known [1]. In the worst case an attacker is able to remotely control the vehicle, which massively endangers the safety of the passengers and other road users. Adding additional WVIs in modern vehicles will lead to more potential threats. Therefore, suitable countermeasures like security standards, security processes and security metrics must be developed and integrated in the development lifecycle of vehicles.

In this paper we propose the DEWI[1] security metric which can be used to perform a structured and detailed system analysis with respect to security. We will also describe the new SAE J3061 standard (Cybersecurity Guidebook for Cyber-Physical Vehicle Systems) and give some insight on how it can be used to evaluate and analyze a WVI. Additionally we will show that the DEWI security metric can be used to support some essential steps defined in the concept phase for the development of CPS described in the new SAE standard.

This paper makes the following contributions:

- DEWI security metric for structured security analysis with comparable and reusable results in different domains.
- DEWI security metric as building block with respect to the new SAE security standard.
- A case study on how to apply the DEWI security metric in combination with the new SAE security standard.

II. Related work

Wireless or remote software update of ECUs (electronic control units) of vehicles is one of the emerging applications today. Remotely updating the software of a vehicle can be very critical with respect to security and safety. Several publications about secure software updates are available. A system where vehicles can be updated using an Internet connection to a portal server is proposed in [3]. The authors mainly focus on the secure data link between the vehicle and the portal. In [4], a security hardware module for ECUs to handle the verification of a new SW update is proposed. The module can also be used for data encryption respectively decryption, digital signatures, and authentication. The wireless interface and the wireless link are not described.

The connection of electric vehicles and the smart grid can also require a wireless interface to the vehicle. In [5]

[1]The ARTEMIS project DEWI (Dependable Embedded Wireless Infrastructure) focuses on the area of wireless sensor / actuator networks and wireless communication. With its four industrial domains (Aeronautics, Automotive, Rail, and Building) and 21 industry-driven use cases, DEWI will provide and demonstrate solutions for wireless seamless connectivity and interoperability in smart infrastructures [2]. (For further details see www.dewi-project.eu)

an authentication protocol for electric vehicles is described. These publications, however, do not include any safety-related aspects or safety analysis of their applications. Additionally, the impact of security on the vehicle safety is not considered.

Several other publications highlight the close relation between safety and security (in automotive applications). In [6] a combined approach of the automotive HARA (hazard analysis and risk assessment) approach with the security domain STRIDE [7] is presented and the impact of security issues on safety concepts at system level are outlined. A combined safety and security development lifecycle is proposed in [8]. Safety and security considerations and activities are introduced in a coordinated way. The authors of [9] investigated the possibility of a combined safety and security approach to standards in the automotive domain. Therefore, existing approaches in the railway and avionics domain with similar challenges are examined and specific requirements for the automotive domain are identified. Contrary to these papers (using an attack-centric approach) our work is based on a system-centric approach allowing the reuse of analyzed components and (sub)systems. The development of guidelines for security in CPS following the well-known safety models and processes is also part of ongoing standardization activities. Currently the SAE released the first security standard for the automotive domain.

A. Security standard for automotive - SAE J3061

SAE recently released a new standard called SAE J3061: Cybersecurity Guidebook for Cyber-Physical Vehicle Systems [10]. The standard describes best practices intended to be flexible, pragmatic, and adaptable in their further application to the vehicle industry as well as to other cyber-physical vehicle systems. The methods and procedures described in SAE J3061 are very similar to the methods and procedures of the well-established functional safety standard ISO 26262 [11]. The characteristic V-model of ISO 26262 is also used in SAE J3061 as the overall cybersecurity (CS) process framework.

The standard defines a complete lifecycle process framework that can be tailored and utilized within each organization's development processes to incorporate CS into cyber-physical vehicle systems from concept phase through production, operation, service, to decommissioning [10]. However, for this paper mainly chapter 6 ("CS process overview") are relevant.

In chapter 6 of SAE J3061 the flow of activities in the concept phase is described. In Figure 1 all steps of the concept phase are shown. For this paper especially the Threat Analysis and Risk Assessment (TARA), the CS Concept, and the functional CS requirements are important.

TARA is used to identify and assess the potential threats to the system and to determine the risk associated with each identified threat. For the highest risk potential threats the so-called CS goals are determined. Once the CS goals are identified for the highest risk potential threats, a CS concept can be developed. Based on the concept and the CS goals the functional (high-level) requirements can be extracted. An iterative process can be used to further refine the results of the single steps (e.g. start with high-level requirements in iteration

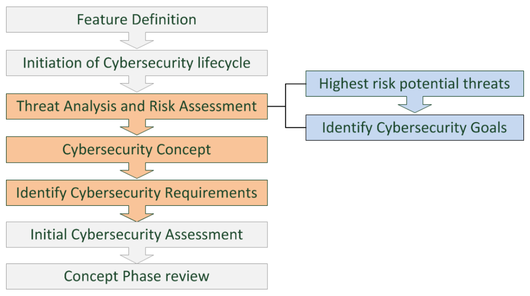

Feature Definition
Initiation of Cybersecurity lifecycle
Threat Analysis and Risk Assessment
Highest risk potential threats
Identify Cybersecurity Goals
Cybersecurity Concept
Identify Cybersecurity Requirements
Initial Cybersecurity Assessment
Concept Phase review

Fig. 1. Flow of activities in concept phase as described in SAE J3061[10]

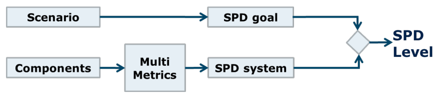


Fig. 2. The required steps of the SHIELD multi-metrics approach

one and refine these requirements in later iterations to obtain technical requirements). The standard provides a high-level and abstract description of the single steps and the expected results. However, no guidance of how to obtain these results is given (except a possible workflow in the appendix).

In this paper we will show that the DEWI security metric described in chapter III can be used to obtain a secure system configuration starting from the CS goals. Furthermore, a CS concept can be created and CS requirements can be extracted based on results of the DEWI security analysis.

B. SHIELD multi-metric approach

SHIELD has been a project co-funded by the EU focused on the research of SPD (Security, Privacy, Dependability) in the context of embedded systems. In the SHIELD project security related prototypes and different metrics for measurable security were developed. For this paper especially the SPD multi-metrics (MM) approach is relevant [12], [13]. The MM approach can be used to evaluate SPD aspects of an entire system. With the SPD metrics a structured system analysis can be performed. It can also be used to find the best system configuration with respect to security. Therefore, the overall score of an entire system is computed by first dividing the system into subsystems and finally into components (in Section III-A descriptions and definitons of the used terminology is provided).

In Figure 2 the required steps of the SHIELD MM approach are outlined. For each scenario an SPD goal value (from 0 for no security up to 100 for highest security level) is defined. After that the system is divided in subsystems and components. This is also illustrated in Figure 3. In the next step system parameters are extracted for each component. Furthermore, a metric consisting of a criticality and a weight value is defined for each system parameter. The MM can now be used to compute the SPD value of the entire system from bottom up (starting at component level). Finally, the SPD value can be compared with the predefined SPD goal. With these steps the overall score of a system can be computed, several system configurations can be compared to each other, and the best fitting system configuration can be chosen (due to performance reasons not necessarily the configuration with the highest score but the one closest to the goal value). The described steps can be applied to perform such an analysis for SPD.

The SHIELD MM approach was successfully applied in DEWI to analyze and evaluate different application scenarios. However some limitations of this approach were identified:

- *Evaluation of dependability*: Dependability is an integrating concept that encompasses the attributes availability (readiness for correct service), reliability (continuity of correct service), safety (absence of catastrophic consequences on the user(s) and the environment), integrity (absence of improper system alterations), and maintainability (ability to undergo modifications and repairs) [14]. Some of the attributes can be contrary: Increasing the reliability and safety of a system (e.g., by maintaining an elevator every day) can decrease the availability of the system (e.g., the elevator can not be used during maintenance). Therefore, using dependability to rate a system or a component can be quite difficult.
- *Privacy as part of security*: In SHIELD, security and privacy are treated as different aspects. However, in many scenarios security and privacy are closely related to each other: If an attacker wants to steal personal data from a Smartphone (privacy aspect), he first has to overcome the lockscreen of the phone or use malicious apps (both security related aspects).
- *Rating of system parameters*: A very important step of the SHIELD MM approach is the definition of a metric (weight and criticality value) for each system parameter. However, no guidance is given on how these values shall be chosen. This leads to incomparable results because security experts will use different (subjective) rating schemes.
- *Choosing the goal values*: The results of the SHIELD MM analysis can be compared to the predefined goals quite easily and based on these results, the best system configuration (e.g., the system configuration with the minimum difference to the goal value) can be chosen. However, choosing the goal values in advance can be difficult (no guidance given, subjective decision) and will again negatively influence the comparability and replicability.

III. DEWI SECURITY METRIC

To overcome the problems stated above we propose the DEWI security metric. After analyzing all security-related requirements of DEWI, a demand for a metric which a) helps the system designers to find the best system configuration, b) can be used to compare different solutions and mechanisms, and c) enables the reuse of security-related components was identified. Therefore, a common understanding about the used metric, the terminology, and common scales to rate system parameters and to define the goal values are needed. The DEWI security metric is still based on the SHIELD MM approach but several adaptions and adjustments were carried out to improve the applicability and the usability of the methodology.

A. Scope and terminology

Contrary to the SHIELD MM approach the DEWI security metric is currently only used to evaluate security aspects (including privacy related aspects as part of security) of the system. Dependability is out of scope for the analysis.

In the following listing the terminology of the DEWI security metric is summarized:

- *System*: the system which will be evaluated using the DEWI security metric.
- *Subsystem*: A system consists of a set of subsystems (logical entities), which are interacting with each other. There can be several nested levels of subsystems.
- *Components*: Components are the smallest building blocks within the DEWI security metric. It is a logical unit (e.g., communication unit) and is part of a subsystem.
- *System parameter*: Each component offers different parameters which can be adjusted (e.g., communication unit: different encryption mechanisms). See also Figure 3.
- *Scenarios*: The same system (e.g., wireless vehicle interface) can be used in different scenarios (e.g., wireless software updates or car sharing applications).
- *System configuration*: In a system configuration each system parameter is adjusted (set to a specific value). Varying these parameter values will lead to several different system configurations.

B. Decomposing the system

In Figure 3 all involved components of an analysis based on the DEWI security metric are shown. By forming logical entities, the system is divided into subsystems and finally into components (an example is given in Section IV). A component contains at least one parameter which can be adjusted. A parameter offers different configuration possibilities. These configuration possibilities can be summarized in a metric. Different system configurations will use different parameter configurations. As illustrated in Figure 4, a system can be used in different scenarios and for each scenario several system configurations can be used. It is also possible that some system configurations cannot be used for certain use cases or domains.

C. DEWI scale to define goal values

One important improvement of the DEWI security metric w.r.t. the underlying SHIELD MM approach is to predefine a goal value for the entire system for each scenario. These values can be compared with the results of the analysis. The goal values can also be used to express which security level is needed for a certain application. For the DEWI security metric a linear scale is used and some characteristic values are defined. For each value we describe the corresponding security level (see Table I). These descriptions are focusing on the attacker abilities and available resources.

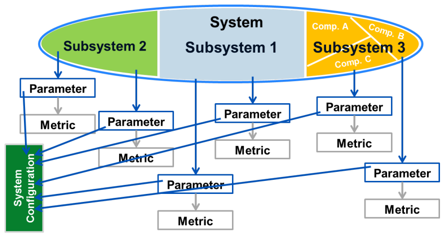


Fig. 3. Decomposing the system into subsystems and components before extracting the system parameters and defining the metric

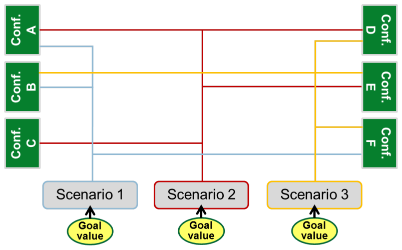


Fig. 4. Different scenarios will require a different security goal value. Several system configurations may fulfill the required goal.

D. DEWI scale to assess system parameters

To assess system parameters, a clear need for defining a common scale was identified in DEWI. The scale will help security experts to choose the score values for the system parameters and to comprehend the choice of their colleagues (when evaluating the previously chosen values). We suggest to use criticality of a system parameter for the assessment instead of the security level (criticality = 100 - security level). We define a non-linear scale for the assessment because it better fits the human interpretation of criticality (illustrated in Table II).

E. DEWI security metric and SAE J3061

The DEWI security metric can be easily integrated in the workflow defined in the new SAE standard (concept phase).

TABLE I
DEWI SCALE TO DEFINE GOAL VALUES: CHARACTERISTIC VALUES AND DESCRIPTIONS

Security level		Description
95	Very strong	Very hard for a team of experts with HW above SotA to break the system (upper practical limit)
75	Good	Experts need time AND access to the vehicle to break the system. No chance for non-experts
50	Average	Experts need time OR access to the vehicle to break the system. Non-experts need time AND access
25	Weak	Non-experts can break the system in affordable time OR with access to the vehicle
5	Very weak	Breaking the system with SotA HW after a short web research possible for every one (lower practical limit)

TABLE II
DEWI SCALE TO ASSESS SYSTEM PARAMETERS

Range	Mode	Description
0-50	normal, operational	System operates in the normal and expected mode. System faults (because of attacks) are unlikely.
50-70	critical	System is still running. Unexpected problems (e.g. unexpected message) can lead to a system fault.
70-100	failure	Behavior of the system is unpredictable and faults (because of attacks) can happen any time.

TABLE III
MAPPING OF THE SAE J3061 CS LEVEL (HEAVENS METHOD [15]) TO THE DEWI SCALE FOR DEFINING GOAL VALUES

DEWI Security level		SAE J3061 Security Level
90		Critical
75	Good	High
50	Average	Medium
25	Weak	Low
10		QM

In the following the required steps are stated and the output of each step is highlighted.

After initiating the CS lifecycle (beyond the scope of this work) the TARA can be performed according to SAE J3061 respectively the HEAVENS [15] method. In this step potential threats are identified and assessed. The output of this step is a list of the highest risk potential threats. This list can be used to identify the CS goals. These goals can now be mapped to the DEWI security goals. The mapping is illustrated in Table III. In the next step the DEWI security metric is used to perform the system analysis and results in a secure system configuration. These system configurations can be used to set up a security concept for the system and to extract the security-related requirements of the system. A very easy way to extract the requirements is to find the minimum system configuration (the configuration which barely fulfills the predefined security goals) and transform the components and the chosen system parameters into requirements.

IV. SECURITY ANALYSIS OF THE WVI: A CASE STUDY

The WVI represents the gateway between the in-vehicle communication system and the outside world. In Figure 5 different types of WVIs are shown. Depending on the application and the design decision of the OEM, the WVI can be part of the central gateway (CGW), which interconnects the different bus systems within a vehicle, or integrated in a dedicated WVI ECU. It is also possible to use a plug-in WVI, which is connected to the in-vehicle communication system via the OBD interface. For business related vehicles (e.g., trucks) a fleet management system with an integrated WVI can be used. Several current vehicles also provide a WVI as part of the infotainment system (with potentially no access to the in-vehicle communication system).

Different upcoming services and applications will require a wireless interface to a vehicle, but especially applications requiring a wireless connection to the outside world as well as access to the in-vehicle communication system (e.g., CAN bus to connect to an ECU) are very critical with respect to

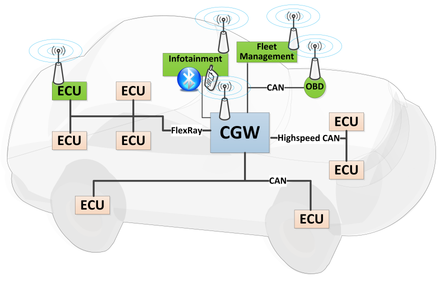


Fig. 5. Possible locations and types of WVIs in a vehicle.

security. In the following listing some promising but also security critical applications and services are presented and described.

- *WVI for remote SW updates*: Required to fix a bug in the SW of an ECU or to enable new features. Today remote updates are not possible (except for Tesla cars). Therefore, vehicle owners have to bring the vehicles to a workshop before new SW can be installed (via a wired connection) on the concerned ECUs.
- *WVI for car sharing application*: The idea is to unlock a vehicle via a Smartphone. Sophisticated security features are needed to ensure that only authorized users can access and use the car.
- *WVI in workshops*: An integrated WVI (part of the in-vehicle communication) can be used for wireless diagnostics and SW updates. However, misuse of the WVI (e.g., unauthorized tuning activities) must not be possible.
- *WVI in electric cars and smart grid*: A (wireless) connection to the smart grid will be required to find the ideal point in time and space to charge the battery of the vehicle.

A. TARA and CS goals

The threat analysis process is used to identify the highest risk potential threats. HEAVENS uses functional use cases to perform the threat analysis and the results are 1) a mapping between identified threats and assets and 2) a mapping between identified threats and security attributes (according to the STRIDE approach [7]). For more information see [15].

To illustrate this step we will use a SW download to an ECU as an example and apply the HEAVENS approach (in a simplified way): The WVI is used to download SW to an ECU of a vehicle (functional use case). A potential threat (T1) is that an attacker acts as a diagnostic tester[2] (DT) and installs malicious SW on an ECU (STRIDE threat: identity spoofing). Another threat (T2) may be that an attacker changes the SW while it is transferred via the wireless link (STRIDE threat: tampering). In both cases the result of the attack can be very critical (e.g., when a safety-critical ECU is reprogrammed).

[2]A tool used in a workshop to connect to the vehicle, run diagnostics and update the SW of ECUs

TABLE IV
TARA RESULTS FOR EXAMPLE THREATS

Threat	Asset	STRIDE threat	Security Attribute	CS goal
T1	Act as DT	Spoofing	Authorization	High
T2	ECU SW	Tampering	Integrity	Medium

TABLE V
SUBSYSTEMS OF THE WVI

Subsystem	Weight	Description	Components
Com. unit	70	Handles wireless traffic, interface to the in-vehicle communication system	Wireless media, Data encryption, Authentication, HW encryption, Bus type
Memory	75	Storing application data, passwords, WVI SW	Normal memory, secure memory
Controller	60	CPU running the WVI SW (microcontroller)	CPU

To calculate the security level (without going into detail in this paper) the threat level and the impact level parameters are used. The result of such a TARA is shown in Table IV and can be mapped to DEWI security goals using Table III.

B. Applying the DEWI security metric

After defining the goal values, the DEWI security metric can be used to analyze the system (the WVI). The following considerations and tables are not meant to be complete but should give an insight of how the DEWI security metric can be used to perform a system analysis. The first step is to split the system into subsystems (see Table V) and components (smallest building blocks). In the next step the components of each subsystem can be identified and a weighting of the subsystem and the components can be performed. A high weight value means that the subsystem/component has a high impact on the security level of the entire system. Finally for each component different system parameters can be identified. Some components of the communication module subsystem can be found in Table VI. The system parameters can now be assessed using the scale defined in Section III-D. After assessing all parameters, different system configurations can be defined (by choosing a system parameter setting per component). Now the SHIELD MM approach (with the formulas defined in [12] and [13]) can be used to compute the security score: First at component, then at subsystem, and finally at system level (bottom up). Different scenarios will lead to different system configurations and therefore different security levels.

The DEWI security metric can be used in an iterative process, were the granularity of the system analysis can be increased in each iteration step. This can be done by adding an additional layer between subsystem and component level. This is also illustrated in Table VII, where the communication unit is grouped in a WiFi module, a bus module, and a communication controller (sub-subsystems).

The resulting system configuration can now be used to create the CS concept for the system. Additionally, it can also be used to extract the security-related requirements of the

TABLE VI
COMPONENTS OF THE COMMUNICATION UNIT SUBSYSTEM (PARTIAL)

Component	Weight	Description	Parameters
Wireless media	65	Used wireless media	WLAN (65) Bluetooth (55) 3G (75)
Wireless data encryption	70	Chosen encryption method for the wireless link	Not used (10) DES 56bit (45) AES 256bit (75)
Wireless authentication protocol	75	Authentication mechanism (wireless)	PSK (50) EAP (70)
HW encryption (Wireless)	35	Wireless HW module supports encryption	No (40) Yes (75)

TABLE VII
SUB-SUBSYSTEM AFTER REFINING AND REGROUPING OF THE COMMUNICATION UNIT SUBSYSTEM

Component	Weight	Description	Parameters
WiFi module	60	HW module handling wireless data exchange	Wireless technology, HW encryption, ...
Bus module	45	HW module handling data exchange with the in-vehicle com. system	Bus protocol, HW encryption, ...
Comm. handler	75	SW module, secure data exchange	Encryption, signatures, integrity checks, authentication, ...

system. As explained in Section III-E, a very efficient way to extract the requirements is to use the minimum system configuration and transform the components (e.g., wireless data encryption) and the chosen system parameter (e.g., AES 256bit) into requirements (e.g., data must be encrypted using AES and a minimum key length of 256bit).

V. CONCLUSION

In this paper we described the DEWI security metric, which can be used to perform a structured security analysis of cyber-physical (automotive) systems. An analysis based on the DEWI security metric will result in secure system configurations covering different scenarios. The proposed methodology can be applied in the automotive domain but can also be used in other domains (e.g. avionics, railway or building domain). Additionally, we illustrated how the DEWI security metric can be used to support the new SAE security standard SAE J3061 (especially the conceptual phase of the security lifecycle). In a case study we showed the steps required to perform the security analysis using the DEWI security metric and considering the steps described in SAE J3061.

In the future we plan to further extend the DEWI security metric to facilitate the extraction of the security concept out of the system configuration. We plan to develop guidelines for a structured approach to support this step. Additionally, the concept phase review step specified in SAE J3061 will be part of future investigations. Our goal is to use the STRIDE approach to evaluate the results of the system analysis using the DEWI security metric.

ACKNOWLEDGMENT

We are particularly grateful to Josef Noll from the SHIELD project for the discussions about the SHIELD MM approach and possible adaptions of it.

The research from DEWI project (www.dewi-project.eu) leading to these results has received funding from the ARTEMIS Joint Undertaking under grant agreement n° 621353. The authors acknowledge the financial support of the COMET K2 - Competence Centres for Excellent Technologies Programme of the Austrian Federal Ministry for Transport, Innovation and Technology (BMVIT), the Austrian Federal Ministry of Science, Research and Economy (BMWFW), the Austrian Research Promotion Agency (FFG), the Province of Styria and the Styrian Business Promotion Agency (SFG).

REFERENCES

[1] Valasek, Chris and Miller, Charlie , "Remote Exploitation of an Unaltered Passenger Vehicle," *White Paper*, p. 93, 2015.
[2] W. Rom, P. Priller, J. Koivusaari, M. Komi, R. Robles, L. Dominguez, J. Rivilla, and W. Van Driel, "DEWI – Wirelessly into the Future," in *Digital System Design (DSD), 2015 Euromicro Conference on*, Aug 2015, pp. 730–739.
[3] D. K. Nilsson and U. E. Larson, "Secure firmware updates over the air in intelligent vehicles," *IEEE International Conference on Communications*, pp. 380–384, 2008.
[4] M. S. Idrees, H. Schweppe, Y. Roudier, M. Wolf, D. Scheuermann, and O. Henniger, "Secure automotive on-board protocols: A case of over-the-air firmware updates," *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 6596 LNCS, pp. 224–238, 2011.
[5] H. Liu, X. Liang, L. Fang, B. Zhang, and J. wen Zhao, "A secure and efficient authentication protocol based on identity based aggregate signature for electric vehicle," in *Wireless Communication and Sensor Network (WCSN), 2014 International Conference on*, Dec 2014, pp. 353–357.
[6] G. Macher, H. Sporer, R. Berlach, E. Armengaud, and C. Kreiner, "Sahara: A security-aware hazard and risk analysis method," in *Design, Automation Test in Europe Conference Exhibition (DATE), 2015*, March 2015, pp. 621–624.
[7] B. Potter, "Microsoft sdl threat modelling tool," *Network Security*, pp. 15–18, 2009.
[8] C. Schmittner, Z. Ma, and E. Schoitsch, "Combined safety and security development lifecylce," in *Industrial Informatics (INDIN), 2015 IEEE 13th International Conference on*, July 2015, pp. 1408–1415.
[9] C. Schmittner and Z. Ma, *Computer Safety, Reliability, and Security: SAFECOMP 2015 Workshops, ASSURE, DECSoS. ISSE, ReSA4CI, and SASSUR, Delft, The Netherlands, September 22, 2015, Proceedings*. Cham: Springer International Publishing, 2015, ch. Towards a Framework for Alignment Between Automotive Safety and Security Standards, pp. 133–143.
[10] SAE, "SAE J3061: SURFACE VEHICLE RECOMMENDED PRACTICE - Cybersecurity Guidebook for Cyber-Physical Vehicle Systems," SAE International, Tech. Rep., 2016.
[11] ISO, "ISO 26262: Road vehicles – Functional safety – Part 1: Vocabulary," ISO, Tech. Rep., 2011.
[12] I. Garitano, S. Fayyad, and J. Noll, "Multi-metrics approach for security, privacy and dependability in embedded systems," *Wirel. Pers. Commun.*, vol. 81, no. 4, pp. 1359–1376, Apr. 2015. [Online]. Available: http://dx.doi.org/10.1007/s11277-015-2478-z
[13] J. Noll, I. Garitano, S. Fayyad, E. Asberg, and H. Abie&, "Measurable security, privacy and dependability in smart grids," *Journal of Cyber Security*, vol. 3, pp. 371–398, 2015.
[14] A. Avizienis, J.-C. Laprie, B. Randell, and C. Landwehr, "Basic concepts and taxonomy of dependable and secure computing," *Dependable and Secure Computing, IEEE Transactions on*, vol. 1, no. 1, pp. 11–33, Jan 2004.
[15] M. I. et al., "Deliverable d2 security models," *HEAVENS Project*, vol. Release 1, no. 1, 2014.

Paper C

M. Steger, C.A. Boano, M. Karner, J. Hillebrand, W. Rom, and K. Römer. **SecUp: Secure and Efficient Wireless Software Updates for Vehicles.** *In Proceedings of the 19th Euromicro Conference on Digital System Design (DSD).*, pages 628–636. Limassol, Cyprus. September 2016.

Summary. This paper presents a structured security analysis of a generic automotive wireless software update system using the DEWI security metrics. The gathered results are then used to define SecUp, a novel security concept enabling efficient and trustworthy wireless software updates for vehicles. In particular, SecUp is tailored to a scenario in which the mechanics in a workshop diagnose and maintain vehicles wirelessly (IEEE 802.11s is utilized as wireless media) using handheld devices. The latter can communicate with the wireless vehicle interface of each surrounding vehicle and verify the presence of new software as well as initiate an update by communicating with a diagnostic tester PC where the new ECU software version is stored. SecUp supports traditional software updates utilizing point-to-point links between the diagnostic equipment and the vehicles as well as advanced update mechanisms such as parallel updates where a new software binary is sent to and installed on several vehicles at the same time. Finally, SecUp is formally evaluated using the STRIDE threat model [48].

My contributions. I am the main author of this paper and carried out the structured security analysis by utilizing the DEWI security metrics, identified the security requirements, as well defined and evaluated the resulting security concept enabling secure as well as efficient wireless automotive software updates. I wrote the vast majority of the paper in collaboration and discussion with the co-authors, who significantly helped to sharpen the description of the security concept. The co-authors also strongly supported the STRIDE-based security concept evaluation.

©2016 IEEE.
ISBN: 978-1-5090-2817-7.
DOI: 10.1109/DSD.2016.11.

Link: http://ieeexplore.ieee.org/document/7723609/.

2016 Euromicro Conference on Digital System Design

SecUp: Secure and Efficient Wireless Software Updates for Vehicles

Marco Steger*, Carlo Boano†, Michael Karner*, Joachim Hillebrand*, Werner Rom*, and Kay Römer†

*Virtual Vehicle Research Center, Graz, Austria
†Institute for Technical Informatics, Graz University of Technology, Graz, Austria
Email: marco.steger@v2c2.at

***Abstract*—Wireless software updates for vehicles are very beneficial for both customers and manufacturers, as they enable performance improvements and error correction without the need of vehicle recalls, as well as a reduction of warranty costs and continuous system upgrades over a vehicle's whole lifetime. However, adding a wireless interface to enable software updates over the air may also expose the vehicle to security threats and make it vulnerable to a variety of attacks. Hence, to protect the safety of the driver and passengers of a vehicle, a strong and comprehensive security concept is needed.**

In this paper, we propose SecUp: a novel security concept enabling efficient and trustworthy wireless software updates for vehicles. SecUp is based on a system-centric structured analysis enabling a secure system configuration. The concept uses, among others, symmetric and asymmetric keys securely stored on the devices in the network to prove the identity of the nodes and to ensure the integrity as well as the confidentiality of data. We evaluate the robustness of SecUp by employing an attacker-centric threat model and show that it is indeed applicable for efficient and trustworthy wireless software updates for vehicles.

I. Introduction

The number of electronic control units (ECU) in a vehicle is rising and the software (SW) installed on these ECUs is becoming increasingly complex, allowing to incorporate new features, but also introducing a growing number of bugs in the automotive SW. Hence, new concepts enabling efficient automotive SW updates are required to support the development and maintenance of modern vehicles.

Current SW update concepts are typically based on inflexible wired point-to-point connections between the vehicle to be updated and a dedicated diagnostic device [1]. Wireless SW updates, similar to firmware-over-the-air (FOTA) updates for smartphones, can offer several advantages, such as an increased flexibility of the SW update process, as well as the possibility of performing updates for several ECUs integrated in various vehicles in parallel. Such parallel SW updates can be beneficial over the whole life-cycle of a modern vehicle (e.g., parallel SW updates is the assembly line or in a workshop). As a result, wireless SW updates can significantly reduce the time needed to maintain modern vehicles (e.g., a mechanic in a workshop can install new SW versions on several vehicles in parallel) and thus will reduce the costs for customers as well as for the OEMs (e.g., by avoiding large vehicle recalls) [1].

Enabling wireless SW updates for vehicles allows a more efficient update process but also requires the introduction of a wireless interface with full access to the in-vehicle communication system (IVCS). Such a wireless vehicle interface (WVI), however, can potentially be exploited by an attacker to endanger the *integrity* of a vehicle by installing malicious SW versions, or to compromise the *privacy* of the driver by eavesdropping user- and vehicle-specific data (see Sect. II for a more comprehensive list of security threats). Indeed, a number of attacks using the wireless interface of the infotainment system of a vehicle is known [2]: in the worst case, an attacker could even remotely control a vehicle, massively endangering the safety of the passengers and other road users. To prevent such attacks and to guarantee the safety of the vehicles and their drivers, a strong and comprehensive security concept protecting the WVI is hence required.

In this paper, we carry out a structured security analysis of a generic automotive wireless SW update system using the SHIELD multi-metric approach [3] and, based on its results, we propose SecUp, a novel security concept enabling efficient and trustworthy wireless SW updates for vehicles. In particular, we tailor SecUp to a scenario in which the mechanics in a workshop diagnose and maintain vehicles wirelessly using handheld devices. The latter can communicate with the WVI of each surrounding vehicle and verify the presence of new SW as well as initiate an update by communicating with a diagnostic tester PC where the new ECU SW version is stored. Handhelds, WVIs, and diagnostic tester PC are interconnected using IEEE 802.11s, which allows to establish a multi-hop wireless mesh network as illustrated in Fig. 1. In an IEEE 802.11s-based network, a new SW version can be installed on several ECUs integrated in different vehicles simultaneously by using a reliable and secure multi-cast communication. Such a parallel SW update process enabling to install SW on several vehicles at the same time can significantly increase the efficiency and productivity in workshops, during vehicle development, as well as in the assembly line.

SecUp describes both the security mechanisms used on the network layer based on IEEE 802.11s, as well as on the application layer of our wireless SW update system. Additionally, it addresses specific requirements and peculiarities of the automotive domain such as support of long life-cycles of modern vehicles, complex and OEM-specific key distribution

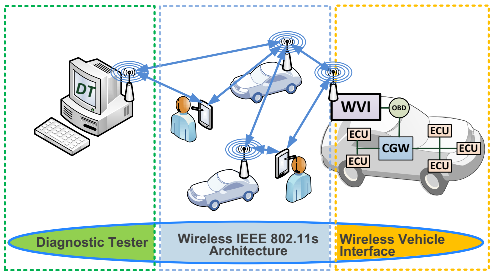

DT
WVI
OBD
ECU
CGW
ECU
ECU
ECU
ECU
Diagnostic Tester
Wireless IEEE 802.11s Architecture
Wireless Vehicle Interface

Fig. 1. The proposed wireless SW update system applied in a typical workshop scenario. System overview with mapping to subsystems.

systems, as well as fleet management systems.

A full description of the scenarios considered in our security analysis and security concept is provided in Sect. III. We describe the performed structured system analysis in Sect. IV and use the resulting secure system configuration as a basis for our SecUp security concept illustrated in Sect. V. Finally, in Sect. VI, we employ an attacker-centric threat model approach based on STRIDE [4] to evaluate the robustness of SecUp and show that it is applicable for the scenario of interest, allowing trustworthy wireless automotive SW updates.

Hence, a key contribution of our work is the introduction of a generic structured work-flow when deriving secure wireless SW updates: first using a system-centric analysis to identify secure system configurations, second extracting the security requirements and defining our SecUp security concept, and finally using an attack-centric threat model to validate SecUp.

II. Related work

Several concepts and systems for over-the-air automotive SW updates have been proposed (e.g., [1], [5]). The authors have specifically focused on secure wireless SW updates and highlighted a number of security threats and aspects, namely:

Vehicle integrity and authentication: As connecting a WVI to a vehicle exposes the latter to a number of attacks, authentication mechanisms must be used to ensure (i) the integrity of the vehicle and (ii) that only authorized entities can access the IVCS. Liu et al. [6] have proposed a vehicle authentication protocol to connect electric vehicles with a smart grid using a WVI. Idrees et al. [7] have also highlighted the need for a secure WVI w.r.t. wireless SW updates.

Data integrity: Data sent through a wireless network can potentially be altered or corrupted by radio interference as well as by a malicious attacker. Especially with respect to secure SW updates, one has to ensure that any (malicious) changes of the transferred data can be detected by the receiver. Mahmud et al. [8] propose a system that ensures data integrity by sending multiple copies of the SW binary. Nilsson et al. [9] use a hash-chain to ensure data integrity in their framework for secure firmware update for intelligent vehicles.

Data confidentiality: A new SW binary that is transferred via the wireless link and installed on the ECU is owned by the OEM or a supplier. An attacker must not be able to obtain it by simply eavesdropping the wireless traffic. The same is true for user- and vehicle-specific data. Idrees et al. [7] have proposed a system that ensures data confidentiality by using dedicated HW modules on every ECU to encrypt/decrypt data.

Key management and exchange: Authentication algorithms and other security mechanisms are based on secret keys. These keys must be securely stored and distributed across the wireless network. Additionally, automotive systems must ensure that an attacker cannot endanger a whole fleet by breaking one vehicle. Idrees et al. [7] have partly addressed this issue by storing keys on a HW Security Module (HSM).

None of the solutions described above fully addresses all the identified security threats and aspects. Nilsson et al. [9], [10] have proposed a system to connect a vehicle with a portal server via an Internet link to install new SW. The authors highlight the related security aspects data integrity and data confidentiality. However, they do not consider the WVI as entry point for a wide variety of attacks (apart from altering the transferred data) and left key distribution, storage and management out of scope. In [8], an architecture for secure SW updates is presented. Although data integrity is ensured by sending multiple copies of the SW, the proposed solution depends on a number of prerequisites and assumptions (e.g., sending multiple copies to ensure secure SW updates) and does not cover aspects such as vehicle integrity and authentication as well as key management and exchange. Idrees et al. [7] have proposed a system for wireless SW updates aiming to address all security aspects described before. In the proposed system, a HSM is used for key management, data encryption and data integrity on the central gateway (the WVI) and on all ECUs of a vehicle. This setup allows data verification directly on the ECUs, but also requires to add a HSM to every ECU, which leads to significant extra costs. Additionally, the proposed system does not take into account the specific type and properties of the wireless link.

Besides not fully addressing all security aspects at once, these solutions are only applicable for wireless point-to-point links. Multi-cast data streams and hence parallel SW updates cannot be realized. The same applies to the Tesla over-the-air SW update solution [11]. Tesla is currently the only OEM providing such wireless remote updates, where a dedicated wireless link (mainly 3G, but a vehicle can also be connected to the users Wi-Fi at home) is used to securely transfer new SW from the manufacturer servers to a vehicle. This point-to-point connection (i.e., between server and vehicle), however, cannot be used to simultaneously install SW in different vehicles and on several ECUs in parallel. Differently from all these works, SecUp, the security concept we propose in Sect. V, supports secure multi-cast data streams and addresses all the aforementioned security aspects at once.

A. Frameworks and Metrics for Security Analysis of Systems

Known frameworks and metrics used to analyze systems w.r.t. security can be classified into (i) system-centric and (ii) attacker-centric approaches. System-centric approaches concentrate on system components and capabilities (e.g., SHIELD multi-metrics approach [3], [12]), whereas attacker-centric approaches consider attacker capabilities, resources and behavior (e.g., Microsoft's STRIDE [4] threat model).

The SHIELD multi-metrics (MM) approach can be used to evaluate security, privacy, and dependability aspects of an entire system first by dividing the system into smaller logical units, second by defining a metric for each unit, and third by computing the security score of the entire system [3], [12]. We use an adapted version of SHIELD for the security analysis.

In the Microsoft STRIDE threat model, a wide variety of potential threats (originally used in the IT security) is collected and grouped into six categories: **S**poofing, **T**ampering, **R**epudiation, **I**nformation disclosure, **D**enial of service, and **E**levation of privilege. To apply STRIDE, first all potential threats on system- as well as on component-level are collected. The resulting list of potential threats can then be used to define proper countermeasures. We will use the STRIDE threat model evaluate SecUp in Sect. VI.

B. Security in IEEE 802.11s Networks

Similarly to [5], we use an IEEE 802.11s network for our automotive SW update system. Steger et al. have highlighted the applicability of IEEE 802.11s as wireless technology for automotive applications requiring a fast, reliable and scalable network, but have not addressed security threats nor provided a security concept.

Security for 802.11s has only been discussed outside of the automotive domain. Tan et al. [13] have described different internal as well as external attacks on IEEE 802.11s and extracted security requirements for systems based on IEEE 802.11s. Using these requirements the authors have performed an evaluation of different approaches and have shown that none of the evaluated algorithms (e.g., [14] and [15]) was able to satisfy the desired security requirements.

Sbeiti et al. [16] use a combination of digital signatures with lightweight authentication trees and symmetric block ciphers called PASER to create a secure IEEE 802.11s mesh network. The proposed system uses the GPS position of the nodes to mitigate a wide range of potential attacks. In [17] the same authors also use different approaches and compared them to PASER, highlighting their inefficiency w.r.t. time (delay of about 70ms per-hop) and power as well as their sensitivity to blackhole and wormhole attacks. However, the use of GPS, as proposed in [16] and [17], inside a workshop building is not applicable for our wireless SW update system.

open11s [18] is an open-source implementation of the IEEE 802.11s standard and it is part of the Linux kernel (kernel version above 3.11). open11s is supporting Simultaneous Authentication of Equals (SAE) [19] as security add-on for IEEE 802.11s. SAE [19], developed especially for 802.11s-based, multi-hop capable mesh networks, is fully integrated in the latest wpa_supplicant version, a free SW implementation of the IEEE 802.11i supplicant on Linux. However, wpa_supplicant and SAE are still under development and currently only support pre-shared keys for IEEE 802.11s networks.

III. Wireless SW Update System for ECUs

In this section we describe the considered wireless SW update system for ECUs. Similarly to [5], we focus on an automotive wireless SW update system based on an IEEE 802.11s multi-hop mesh network, as it is applicable for different application scenarios: (i) a typical workshop scenario where several vehicles located inside and around the workshop building are repaired and maintained (e.g., receiving new SW for several ECUs) by mechanics at the same time (Fig. 1); (ii) vehicle development, where development engineers have to update the SW of one or more ECUs several times to evaluate and test newly developed features; (iii) the use in the assembly line, where the SW of many or all ECUs of the vehicle is initially installed or updated to the latest version; and (iv) remote updates from home, where new SW can be installed on the ECU of a vehicle without the need of physically bringing it to the workshop.

Depending on the application scenario, the wireless SW update is performed by different users with different expertise (development engineers as expert users, mechanics as trained users, and, in case of remote updates, vehicle owners as untrained user) in different environments. Therefore, the considered wireless SW update system must fulfill different security requirements and address different security aspects depending on the considered application scenario. In Sect. IV we propose a structured security analysis that can be used to address different application scenarios at once and results in a secure system configuration. Additionally, we take into account that a modern vehicle consists of different types of ECUs with, in case of a failure, different impact on vehicle safety. To address this in a structured way we classify ECUs accordingly:

- **Uncritical - Class 1:** Mainly ECUs related to entertainment/infotainment. Hardly any effect on vehicle safety and performance. Lowest criticality level.
- **Body and comfort - Class 2:** All ECUs related to the body CANs of a vehicle (e.g., window lift or heating, ventilation and air conditioning). No (direct) influence on vehicle dynamics or driving. Medium criticality level.
- **Powertrain, chassis, driver assistance - Class 3:** Complex and safety-critical systems that control safety features of a vehicle and have strong impact on vehicle dynamics and driving. (Very) high criticality level.

Please note that the security concept presented in Sect. V mainly focuses on the security threats, aspects, and requirements with respect to a typical workshop scenario. Dedicated security concepts for the other scenarios, however, will apply similar security mechanisms and will be defined following the ideas presented in Sect. V.

SecUp uses an IEEE 802.11s mesh network to interconnect the vehicles (via the WVI), the diagnostic tester (DT), and different handheld devices such as smartphones and tablets. In such a mesh network, data can be directly exchanged between two nodes but can also be forwarded by some (relay) nodes in a multi-hop network. For a typical workshop scenario as shown in Fig. 1, data (e.g., a new SW version to be installed on one or more vehicles), which shall be sent from the DT to a vehicle in the workshop, can be forwarded by other vehicles to reach its final destination. The multi-hop capability of IEEE 802.11s guarantees a good wireless coverage and simplifies the extension of the wireless network. Additionally, the data can be routed through the network by using different paths (i.e., forwarded by different nodes). The resulting redundancy increases the reliability of the network. A IEEE 802.11s network also supports multi-cast data stream, that can be exploited to realize parallel SW updates, where a SW binary first is transferred to the WVIs of several vehicles in parallel and second installed on the concerned ECUs. In [20] our framework enabling secure and efficient automotive wireless SW updates is also described in detail and more insight on the involved nodes, the interconnections between these nodes, the benefits of parallel SW updates and the users of the wireless SW update system is given.

To secure the wireless IEEE 802.11s network in a lightweight way, we utilize wpa_supplicant, a generic security framework for different types of wireless networks and SAE [19], developed especially for 802.11s-based, multi-hop capable mesh networks. SAE is fully integrated in the latest wpa_supplicant version and thus can be used to secure our wireless IEEE 802.11s mesh network.

SAE offers an authentication mechanism based on pre-shared keys. Using pre-shared keys means to have the same key on every wireless node. For example, the same key is used on a whole fleet of vehicles: by breaking one node the pre-shared key, used for a whole fleet of vehicles, may be extracted. Because of that, pre-shared keys must be used with caution and applying security only on the network layer is not enough to guarantee secure wireless SW updates. Therefore a cross-layer solution is required in the final security concept using the open11s security features as first layer of security.

IV. Security analysis of the SW update system

In this section we present a structured system analysis that can be used to address all aforementioned application scenarios and ECU types. We perform a system-centric security analysis as basis for the security concept and to evaluate the resulting security concept using an attack-centric approach (Sect. VI). Thereby our approach can also be used to support the new cybersecurity standard SAE-J3061 [21]. SAE-J3061 is closely following the well known and wide spread functional safety standard ISO 262626 [22] and is covering the whole lifecycle of a vehicle. The standard was mainly defined for the automotive domain, but it can also be applied in other industries. In [23] we described in more detail how the structured system analysis can be used to support SAE-J3061.

The structured system analysis presented in this paper is an adapted version of the SHIELD MM approach [3], [12]. The latter is typically used to evaluate security, privacy, and dependability aspects of an entire system. Contrary to SHIELD, we treat privacy aspects as part of security instead of analyzing both of them individually. Additionally, dependability is out of scope for our security analysis and we propose instead a linear scale to support users when applying the security analysis (Sect. IV-A).

The analysis is performed in three main steps. First a security goal value on the above scale must be defined for every application and every ECU class. This goal value will be used at a later stage to compare it with the computed score values and thereby to find the best system configuration. It is important to note that the configuration with the highest security score does not necessarily have to be the best choice: adding strong security mechanisms may – beside costs – lead to significant delays in the system that may, in the worst case, permanently impair its functionality.

In the second step (i) the system is divided into subsystems and into components, (ii) a weighting of the subsystems and the components is performed, and (iii) for every component feasible system parameters are identified. A system parameter of a component can be seen as one possible way to configure the component. An example component would be the data en- and decryption unit. Possible system parameters for this example component could be different encryption mechanisms and different key lengths. By varying the system parameters, different system configurations can be defined. Note that only meaningful and realistic configurations shall be used. There is no benefit in computing the system score for any possible system configuration.

In the third step the security score of the entire system is computed by first calculating the score for each subsystem and finally for the entire system. For this purpose, the equations defined as part of the SHIELD MMs are used [3]. Instead of directly using the security value (S_i) of a component, the equation is based on the criticality (C_i) of the component:

$$C = \sqrt{\sum_i \frac{c_i^2 * w_i}{\sum_i w_i}} \quad with \tag{1}$$

$$w_i = \left(\frac{W_i}{max(W_i)}\right)^2 \quad and \quad S_i = 100 - C_i \tag{2}$$

with C being the resulting criticality value of the analyzed (sub)system, c_i the criticality of each of the components of the subsystem, W_i the weight values of the components (between 0 and 100), and w_i the normalized weight values (between 0 and 1). The resulting values can now be compared with the predefined goal values and the best system configuration can be chosen (e.g., the configuration closest to the goal value).

A. Defining the Security Goal Values

The security goal value is defined within a range from 0, meaning no security, up to 100, which can be expressed as “hundred percent secure” (both theoretical limits). Following the guidelines presented in the SHIELD publications [3], [12], a security expert performing the SHIELD analysis has to choose the security goal values for each considered scenario upfront. The defined goal values are compared with the results of the subsequently performed analysis and these goal values can also be used to express which security level is needed for a certain application. However, no insight is given in [3]

TABLE I. ECU classification and security goal value for each scenario according to the impact on vehicle safety

	Assembly line	Vehicle development	Workshop	Remote update
Class 1	50	60	50	75
Class 2	55	60	60	80
Class 3	60	60	70	90

TABLE II. Weights of the wireless SW update subsystems

Subsystem	W_i (0-100)	w_i(0-1)
WVI	70	1
Wireless architecture	60	0.73
Diagnostic tester	50	0.51

or [12] on how these values shall be chosen. This leads to incomparable results because security experts will use different (subjective) rating schemes. One important improvement of our adapted security analysis w.r.t. the underlying SHIELD MM approach is that we provide a scale which can be used to predefine the security goals more easily. Therefore, a linear scale (0-100) is used and some characteristic security levels SL are defined (e.g., very weak equals SL=5, average equals SL=50 and good security equals SL=75). For each value we describe the attacker abilities and available resources and map it to a certain security level (e.g., good security: experts need time and access to the vehicle to break the system).

We also use this scale to choose the goal values for the four application scenarios (wireless SW updates in a typical workshop scenario, during vehicle development, in the assembly line, or remotely from home) and the three different ECU classes. The resulting security goal values are presented in Table I. The remote update scenario requires the highest security level because the update is done in public by untrained users. SW updates in workshops or in the assembly line are performed by trained users in a (highly) secure environment and therefore the required level is lower compared to the remote scenario. In the vehicle development phase the main threat is industrial espionage, where an attacker tries to eavesdrop the currently developed SW when transferred and installed over the air. Never the less, independently from the ECU class, we have to ensure that this is not possible at all.

B. Decomposing the Wireless SW Update System

The considered wireless SW update system can be divided into three subsystems (see Fig. 1 and Table II) and a weighting of the subsystems is performed. The weight values are in a range from 0, (meaning not relevant for security), and 100, (very relevant). As shown in Table II, the weight values are normalized (between 0 and 1) for further calculations using Eq. 2. Guidelines and an example of how to choose the weight values are presented in [3].

In the next step the subsystems are further divided into components. In Table III one component of the WVI subsystem is presented. The component describes the type of WVI w.r.t. the connection of the WVI with the in-vehicle communication system. For the workshop scenario a plug-in solution (e.g., using the OBD interface of a vehicle) for the WVI is more suitable because it guarantees that also older vehicles can be maintained. However, a plug-in WVI is more critical w.r.t.

TABLE III. *WVI type* component and possible parameters

Component / Parameter	S_i	Description
WVI type		Either a plug-in solution (e.g., via OBD) or fully integrated in the IVCS (e.g., as ECU)
Integrated	85	WVI is fully integrated and cannot be removed easily
Plug-in with trusted DT	75	The WVI can only be used in connection to an trusted DT (e.g., using certificates)
Plug-in location	75	The WVI can only be used in a certain area or building (e.g., only within a workshop)
Plug-in normal	45	The WVI can be used without any additional security checks

security than a fully integrated WVI solution (e.g., a WVI can be stolen and hijacked by an attacker). Additional security checks using location information (e.g., a WVI can only be used when inside a workshop building) can be applied to make the WVI plug-in solutions more secure. These possible variations are the system parameters of the component.

C. System Configuration and Security Requirements

A system configuration contains all identified components with a chosen system parameter for each component. Based on this set of system parameters, first the security score of each subsystem is calculated and then the subsystem scores are used to compute the security score of the entire system. Sample configurations (non-exhaustive list of possible WVI components and parameters) for the workshop as well as the remote update scenario are presented in Table IV.

The system parameters are chosen w.r.t. the functional requirements of the specific application scenarios: the workshop scenario requires to maintain different types, brands and ages of vehicles (backward compatibility as key factor) and a plug-in WVI must hence be used. Contrary to that, remote SW updates are only relevant for modern/future vehicles and therefore only a fully integrated WVI is considered. Such considerations are applied for every component and system parameter and a few reasonable system configurations can be identified per application scenario. For each of these system configurations first the scores of the subsystems are computed. Based on that, the entire system security score can be calculated by combining the results of the WVI subsystem (see Table IV for a sample configuration) with the computed results of the other subsystems. The security score is computed for all reasonable system configurations and finally one system configuration is chosen for each application scenario. We suggest to take the system configuration closest to the goal value.

The final system configurations can now be used to extract the security requirements and in further consequence to start defining the security concept. The subsystem configuration presented in Table IV can be converted into security requirements by analysing and combining the components and the corresponding system parameters:

- **Req.1**: *The user must have physical access to the (inside of the) vehicle to authorize the WVI.* In the workshop scenario this requirement is fulfilled by plugging in the WVI.

TABLE IV. SAMPLE CONFIGURATION OF THE WVI SUBSYSTEM FOR THE WORKSHOP AND THE REMOTE SOFTWARE UPDATE SCENARIO

Component	W_i	Workshop		Remote	
		Parameter	S_i	Parameter	S_i
WVI type	85	Plug-in trusted	75	Integrated	85
Key Storage	70	Secure Memory	80	HSM	90
Authorization	60	Physical Access	75	Physical Access	75
Subsystem score			76		83

- **Req.2**: *All required keys must be stored in a dedicated and secure memory (workshop scenario); remote: using a HW security module (HSM).*
- **Req.3** (for the workshop scenario only): *The WVI must only connect to trusted DTs (e.g., using public key infrastructure (PKI) enabling secure digital signatures).*

V. OVERVIEW OF THE SECUP SECURITY CONCEPT

SecUp is based on the secure system configuration resulting from the above analysis and addresses all security aspects stated in Sect. II as well as the security-related requirements in Sect. IV-C (extracted from the secure system configuration).

A. *Key Management and Exchange*

As illustrated in Fig. 2, our security concept for trustworthy wireless software updates for vehicles is using different keys and security mechanisms. In the following these keys are summarized and their usage is described:

Master key pair (RSA or ECC): The master key pair (consisting of a private and a public key) is mainly used to handle the authentication between the DT and the WVIs. Additionally, it will be used to exchange the session key (required for multi-cast data streams and hence enabling secure parallel SW updates) and to exchange critical, vehicle-specific data (e.g., seed and corresponding key needed to authorize a SW update on ECU level).

Session key (AES): This key will be used to encrypt multi-cast data (e.g., a SW binary sent to several vehicles).

Network key: This key is needed to interconnect all nodes using the wireless IEEE 802.11s network. As mentioned previously the network authentication is based on SAE [19] and network security as well as network authentication can be seen as the fundamental security level.

ECU authorization key: To unlock an ECU (authorization step) typically a seed & key mechanism is used: The ECU creates a seed (e.g., a 4 byte random number), sends the seed to the WVI and internally computes the corresponding ECU-key using a secret algorithm. Meanwhile, the WVI will forward the seed to the DT (encrypted, using the public master key of the DT). The DT will decrypt the message, compute the ECU-key using the received seed, and send the encrypted ECU-key back to the WVI. After decryption, the WVI will forward the ECU-key to the ECU and the latter will compare the received ECU-key with the internally computed key. The WVI is now authorized to install new SW on the ECU.

User PIN: To use the DT (the wireless update system), a user will have to authenticate himself directly on the DT or

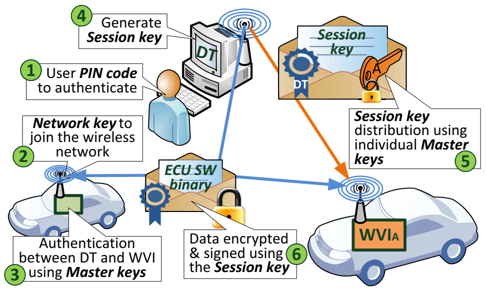

4 Generate Session key
1 User PIN code to authenticate
DT
Session key
DT
A
Session key distribution using individual Master keys 5
Network key to join the wireless network 2
ECU SW binary
WVIA
Authentication between DT and WVI 3 using Master keys
Data encrypted & signed using 6 the Session key

Fig. 2. Usage of different keys in the security concept. Parallel updates, where an ECU SW binary is installed on both vehicles simultaneously, can be performed using a secure, IEEE 802.11s based multi-cast.

on a WVI by using a smartcard and entering a user-specific PIN code. Different user profiles will be used to distinguish between different user roles (e.g., in a workshop some users will only be allowed to run wireless diagnostics, while others will also be allowed to perform wireless SW updates).

In a typical workshop scenario, where a mechanic wants to maintain a vehicle (e.g., running wireless diagnostics and installing new ECU SW), he will first have to authenticate himself on the DT (Fig. 2, step ①) using the user PIN and connect the WVI with the vehicle using the OBD interface, usually located in the cabin of a vehicle. Thereby, Req.1 (*The user must have physical access to the (inside of the) vehicle to authorize the WVI*) is fulfilled.

After connecting the vehicle and the WVI, the latter will power up and join the wireless network (Fig. 2, step ②) using the network key. The IEEE 802.11s network is secured using SAE. The latter is based on pre-shared keys (same key on every node) and hence cannot be considered as fully secure. Therefore, additional authentication and encryption mechanisms are applied on the application layer.

In the next step, the WVI will establish a secure connection to the DT (Fig. 2, step ③). Following the requirement Req.3 (*The WVI must only connect to trusted DTs*) defined in Sect. IV-C, the authentication phase (on application layer) between the WVI and the DT is based on PKI. This means that a key pair consisting of a private and a public key (asymmetric keys) is used on every device ensuring an unambiguous identification between all nodes without using the same (symmetric) key on all WVIs respectively on all vehicles. The asymmetric keys (e.g., based on RSA or ECC) are used as "master key pair" in our concept and are stored on a secure memory (following Req.2: *All required keys must be stored in a dedicated and secure memory*).

To exchange the public master keys, a secure pairing of the nodes must be performed. In this phase the own (generated) key pair as well as the public master key of the other device must be securely stored on the devices (the WVI stores the public master key of the DT and vice versa). This is required and important to prove the identity of the devices later on. Note that the exchange of the public master keys is critical and can only be done by authorized users (before a WVI can be used properly).

A new and unpaired WVI will initially boot in a specific mode, where it creates an IEEE 802.11s network with a random key, which is created in the production phase and shipped with the WVI. An authorized user will use this key to interconnect the new WVI with the DT (on network layer, IEEE 802.11s) and furthermore to handle the initial master key generation as well as distribution: after connecting to the WVI, another onetime password can be used to trigger the computation of the master key pair on a WVI, to store the public master key of the DT in the secure memory of the WVI, and to share the public master key of the WVI with the DT.

Alternatively, the master key pair is already computed when the WVI is produced. A onetime password can be used to initially exchange the keys (public master key and network key) via a wired connection between DT and WVI (e.g., using a USB connection). The DT and the WVIs (one by one, every WVI will have its own key pair) will exchange the public master keys and store them securely.

In [24] the security levels of symmetric and asymmetric keys are compared: Asymmetric keys equivalent to AES-256bit symmetric keys require either RSA key pairs with 15360bit (which is computationally infeasible in embedded systems today) or ECC key pairs with 512bit key length. Therefore, the use of asymmetric keys offering a similar security level (e.g., to encrypt the binary before sending via the wireless network), is significantly more expensive with respect to (computing) power and time [24] and shall only be used when necessary. Additionally, to enable efficient as well as secure multi-cast data streams the same (symmetric) key must be used on all concerned devices.

Because of that, we additionally use symmetric keys (AES, 256bit) as "session keys" besides the asymmetric master key in our concept. A session key is generated by the DT (Fig. 2, step ④) and encrypted using the public master Key of the concerned WVIs (which shall be part of the session, e.g., to receive the same SW binary). Finally, the encrypted session key is distributed to the WVIs using the wireless IEEE 802.11s network (Fig. 2, step ⑤). Each WVI can now use its private master key to decrypt the session key.

To securely and efficiently exchange data between the DT and one or more WVIs, the symmetric, AES-based session key is used (Fig. 2, step ⑥).

B. Vehicle Integrity and Authentication

Authentication between the WVI and the DT can be achieved by first connecting to the wireless network using the pre-shared network key and by then using the master keys of the WVI and the DT.

Once the WVI has joined the wireless network, it will listen for DT beacons. The beacon informs the WVI about the presence of the DT and the available services. The WVI will then try to establish a secure connection with the DT (sending its digital signature) and the DT will answer (after proving the identity of the WVI) with a message containing its digital signature and the encrypted session key. The WVI can then check the identity of the DT by using the stored public master key of the DT. After approving the identity of the DT, a secure connection between the DT and the WVI is established.

In our concept we propose to use a timeout for the session key: If the session is inactive (e.g., lunch break in a workshop but session still established) for a predefined period (e.g., 1h), a re-authentication is required. Additionally, a maximum validity period (e.g., one working day or 12h) for a session key shall be used due to security reasons.

Based on the predefined user profiles, an authenticated user can use the secure connection between the DT and the WVI to run wireless diagnostics (standard user) as well as to install new SW on the vehicle (empowered user, verified by the DT).

C. Data Integrity

To ensure the integrity of the transferred data, Hash-based Message Authentication Codes (HMAC) or Cipher-based Message Authentication Codes (CMAC) can be used. In our security concept we propose to compute the HMAC/CMAC for every message using the session key.

Additionally, after the whole SW binary has been transferred from the DT to the WVI(s), the DT will compute the hash of the entire SW binary, encrypt it using the public master key of (every) WVI, sign the message using its private master key, and finally send it to the WVI. The WVI will use the locally and securely stored public master key of the DT to prove that the message was signed by the DT, and its own master key to encrypt the message (to extract the hash of the SW binary). Finally the WVI can now ensure that the entire SW binary was not altered while transferred via the wireless network.

D. Data Confidentiality

The confidentiality of the transferred data can be ensured by encrypting the data using the session key (e.g., encryption of the SW binary) or the public master key of the receiving node (e.g., to encrypt the session key).

The session key is generated on the DT and distributed to all concerned WVIs via the wireless network. After the distribution of the session key, it can be used to encrypt/decrypt multi-cast data.

VI. Security Concept Evaluation using STRIDE

The STRIDE approach can be used to analyze the security of a system (attack-centric approach, see Sect. II-A) by identifying a number of potential security threats and grouping them into six categories: **S**poofing, **T**ampering, **R**epudiation, **I**nformation disclosure, **D**enial of service, and **E**levation of privilege. To apply STRIDE, each part of the system needs to be analyzed and all potential threats that fall into the STRIDE categories, are determined for every component or process. Based on these threats, suitable countermeasures can be defined.

This approach, however, can also be used to evaluate a security *concept* of a system. In this paper we propose to use the attack-centric STRIDE approach to evaluate SecUp, which was defined based on the results of the structured, system-centric security analysis described in Sect. IV. Due to space restrictions we do not to present the complete evaluation, but specifically focus on two groups of the STRIDE threat model and thereby on the three most critical and challenging security threats. In the following, these threats are described, possible consequences are stated, and relevant countermeasures w.r.t. SecUp are presented.

A. STRIDE Threats: Spoofing Identity & Tampering

The concept of spoofing identity allows unauthorized (and malicious) nodes/devices/users/code to use the identity of authorized entities and hence their security credentials. With respect to secure wireless SW updates, we identified identity spoofing of the WVI or the DT as the most critical threats.

By spoofing the identity of the WVI (Threat T1), an attacker can indeed potentially try to register for new SW updates. In such a case, the malicious WVI will connect to the DT, request new SW versions, receive these SW binaries and store them for later (unauthorized) SW updates (e.g., tuning or activating fee-based features of a vehicle).

Even more critical threats can arise if an attacker successfully spoofs the identity of a DT (Threat T2): The malicious DT can connect to a WVI (already plugged-in to a vehicle) and use the connection to the vehicle to (i) read vehicle- and driver-specific data (endangering the privacy of the user) or to (ii) install new malicious SW on ECUs of the vehicle (endangering the safety of the driver and other road users).

Another way to install malicious SW on a vehicle is to alter the data while the latter is transferred over the air from the DT to the target WVI (Threat T3). An attacker first breaks the network level authentication to join the wireless network and then applies a path diversion attack [13] can be applied to ensure that all packets are routed via the malicious node. As a result, an attacker is then able to start tampering with the transferred data.

B. Analyzing the identified Threats w.r.t. SecUp

The identified and considered STRIDE threats are summarized in Table V. To successfully perform an attack following threats T1-T3, an attacker has to first overcome the authentication step on the network level; i.e., breaking the SAE-based security mechanism (to the best of our knowledge, however, no successful attacks exist today) or find out the pre-shared network key. For the latter, an attacker can try to extract the network key from an already paired WVI. The keys, however, are stored in a secure memory and extracting them requires significant expertise, effort, and time.

We now describe how potential attacks w.r.t. threats T1-T3 can look like, in the unlikely case that an attacker is able to overcome the security mechanisms on the network level and the corresponding SecUp countermeasures. The identified threats, possible consequences, and the SecUp countermeasures are also summarized in Table V.

Thread T1 & T2: An attacker can try to connect to a DT by spoofing the identity of a WVI (T1) or to connect to a WVI by spoofing the identity of a DT (T2). Therefore, an attacker would need to fake the digital signature of the malicious node (acting either like a WVI or DT). In our security concept strong authentication mechanisms based on RSA or ECC signatures are used. These mechanisms are considered as very secure [24] and therefore breaking the system within a reasonable time is not possible (an attacker will not stay unnoticed while running such an attack within a workshop). An attacker can also try to pair the malicious WVI and the DT. However, only privileged users (e.g., the head of a workshop) are allowed to do that and physical access to the DT is hence required.

Thread T3: To successfully install malicious SW on an ECU by altering the transferred data, an attacker has to first ensure that all data packets are forwarded by the malicious device (e.g., realized using a path diversion attack [13]). The malicious node can then alter the data before forwarding the data packet.

According to our concept, HMAC or CMAC is applied on every data message to ensure the integrity of the transferred data. To overcome that, an attacker needs to find a way to break the HMAC/CMAC or to figure out the AES 256bit Session key (stored in secure memory on all devices). Additionally, after transferring the entire SW binary to the target WVI(s), the DT will compute the hash of the entire SW binary, encrypt it as well as sign it using the master keys of the DT/WVI, and send this to the WVI. Practically, this means that an attacker will additionally need to break the RSA/ECC-based digital signatures. Today, no attacks to break such digital signatures in reasonable time are known.

As a result of the performed STRIDE evaluation focused on spoofing and tampering, we can state that an attacker following T1, T2 or T3 will not have a realistic chance to break the system in a reasonable time.

VII. Conclusion

In this paper we propose SecUp, a security concept for efficient and trustworthy wireless software updates for vehicles. SecUp is based on the results of the presented structured system analysis, which can be used to analyze several application scenarios (e.g., wireless SW updates performed in a workshop, in the vehicle development or in a assembly line, as well as remote SW updates from home) at once and results in secure system configurations. These secure system configurations are used to extract the related security requirements and finally to define SecUp.

We evaluated SecUp using the STRIDE threat model and thereby proved its applicability for efficient and trustworthy wireless automotive SW updates.

We are currently working on a secure HW prototype of the WVI. That will be used to evaluate the impact of different security mechanisms such as RSA vs. ECC or HMAC vs.

TABLE V. IDENTIFIED THREATS, POSSIBLE CONSEQUENCES AND SECUP COUNTERMEASURES

Threat	Threat description	Consequences	SecUp countermeasure
Threat T1	Spoofing the identity of a WVI	Steal a SW binary; Unauthorized SW updates	Authentication on network (SAE) and application layer (RSA, ECC); secure pairing (physical access to DT required)
Threat T2	Spoofing the identity of the DT	Exploit connected WVI: Read user-specific data, install malicious SW	Authentication on network (SAE) and application layer (RSA, ECC); insider information required to overcome ECU seed & key
Threat T3	Altering the SW while being transferred to the WVI	Negatively impair the wireless link; worst case: install malicious SW	Authentication on network (SAE) and application layer (RSA, ECC); Data encryption (AES); Data integrity: HMAC or CMAC per packet plus hash of SW binary after complete data transfer

CMAC on system performance. Additionally, we are planning to compare the performance of different security functions implemented in SW (e.g., a Java security library) and HW (e.g., using a Trusted Platform Module chip). Furthermore we will evaluate the impact of SAE authentication and encryption (security on network level) on the performance (e.g. of the wireless IEEE 802.11s network utilizing a real IEEE 802.11s testbed consisting of 12 network nodes.

ACKNOWLEDGMENT

We are particularly grateful to Josef Noll from the SHIELD project for the discussions about the SHIELD MM approach and possible adaptions of it. Also we want to thank our colleagues from Volvo Trucks Dhasarathy (Dhas) Parthasarathy and Jonas Hagerskans for providing us an ECU to test and demonstrate the SW update process using real hardware.

The research from DEWI project (www.dewi-project.eu) leading to these results has received funding from the ARTEMIS Joint Undertaking under grant agreement n° 621353. The authors acknowledge the financial support of the COMET K2 - Competence Centres for Excellent Technologies Programme of the Austrian Federal Ministry for Transport, Innovation and Technology (BMVIT), the Austrian Federal Ministry of Science, Research and Economy (BMWFW), the Austrian Research Promotion Agency (FFG), the Province of Styria and the Styrian Business Promotion Agency (SFG).

REFERENCES

[1] Redbend Software, "Updating Car ECUs Over-The-Air (FOTA)," *White Paper*, pp. 1–14, 2011.

[2] Valasek, Chris and Miller, Charlie , "Remote Exploitation of an Unaltered Passenger Vehicle," *White Paper*, p. 93, 2015.

[3] I. Garitano, S. Fayyad, and J. Noll, "Multi-metrics approach for security, privacy and dependability in embedded systems," *Wirel. Pers. Commun.*, vol. 81, no. 4, pp. 1359–1376, Apr. 2015.

[4] B. Potter, "Microsoft sdl threat modelling tool," *Network Security*, pp. 15–18, 2009.

[5] M. Steger, M. Karner, J. Hillebrand, W. Rom, E. Armengaud, M. Hansson, C. A. Boano, and K. Romer, "Applicability of ieee 802.11s for automotive wireless software updates," in *Telecommunications (ConTEL), 2015 13th International Conference on*, July 2015, pp. 1–8.

[6] H. Liu, X. Liang, L. Fang, B. Zhang, and J. wen Zhao, "A secure and efficient authentication protocol based on identity based aggregate signature for electric vehicle," in *Wireless Communication and Sensor Network (WCSN), 2014 International Conference on*, Dec 2014.

[7] M. S. Idrees, H. Schweppe, Y. Roudier, M. Wolf, D. Scheuermann, and O. Henniger, "Secure automotive on-board protocols: A case of over-the-air firmware updates," *Lecture Notes in Computer Science*, vol. 6596 LNCS, pp. 224–238, 2011.

[8] S. M. Mahmud, S. Shanker, and I. Hossain, "Secure software upload in an intelligent vehicle via wireless communication links," in *Intelligent Vehicles Symposium, 2005. Proceedings. IEEE*, June 2005, pp. 588–593.

[9] D. K. Nilsson and U. E. Larson, "Secure firmware updates over the air in intelligent vehicles," *IEEE International Conference on Communications*, pp. 380–384, 2008.

[10] D. Nilsson, P. Phung, and U. E. Larson, "Vehicle ECU classification based on safety-security characteristics," *Road Transport Information and Control - RTIC 2008 and ITS United Kingdom Members' Conference, IET*, pp. 1–7, 2008.

[11] N. Gabe, "Over-the-air updates on varied paths," *Automotive News*, 2016-01-25.

[12] J. Noll, I. Garitano, S. Fayyad, E. Asberg, and H. Abie&, "Measurable security, privacy and dependability in smart grids," *Journal of Cyber Security*, vol. 3, pp. 371–398, 2015.

[13] W. K. Tan, S.-G. Lee, J. H. Lam, and S.-M. Yoo, "A security analysis of the 802.11s wireless mesh network routing protocol and its secure routing protocols," *Sensors*, vol. 13, no. 9, p. 11553, 2013.

[14] M. S. Islam, M. A. Hamid, and C. S. Hong, "SHWMP: A Secure Hybrid Wireless Mesh Protocol for IEEE 802.11s Wireless Mesh Networks," in *Transactions on Computational Science VI*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 95–114.

[15] J. Ben-Othman and Y. I. Saavedra Benitez, "IBC-HWMP: a novel secure identity-based cryptography-based scheme for Hybrid Wireless Mesh Protocol for IEEE 802.11s," *Concurrency and Computation: Practice and Experience*, vol. 25, no. 5, pp. 686–700, 2013.

[16] M. Sbeiti, A. Wolff, C. Wietfeld, and I. Technology, "PASER: Position Aware Secure and Efficient Route Discovery Protocol for Wireless Mesh Networks," in *International Conference on Emerging Security Information, Systems and Technologies - SECURWARE*, no. c, 2011.

[17] M. Sbeiti and C. Wietfeld, "One stone two birds: On the security and routing in wireless mesh networks," in *Wireless Communications and Networking Conference (WCNC), 2014 IEEE*, April 2014.

[18] "open80211s – an open-source implementation of the recently ratified ieee 802.11s wireless mesh standard," https://github.com/o11s/open80211s/wiki/HOWTO.

[19] D. Harkins, "Simultaneous authentication of equals: A secure, password-based key exchange for mesh networks," in *Sensor Technologies and Applications, 2008. SENSORCOMM '08. Second International Conference on*, Aug 2008, pp. 839–844.

[20] M. Steger, M. Karner, J. Hillebrand, W. Rom, C. Boano, and K. Roemer, "Generic Framework Enabling Secure and Efficient Automotive Wireless SW Updates," in *IEEE ETFA 2016 – 21st IEEE International Conference on Emerging Technologies and Factory Automation*, 2016, pp. 1–8.

[21] SAE, "SAE J3061: SURFACE VEHICLE RECOMMENDED PRACTICE - Cybersecurity Guidebook for Cyber-Physical Vehicle Systems," SAE International, Tech. Rep., 2016.

[22] ISO, "ISO 26262: Road vehicles – Functional safety – Part 1: Vocabulary," ISO, Tech. Rep., 2011.

[23] M. Steger, M. Karner, J. Hillebrand, W. Rom, and K. Roemer, "A Security Metric for Structured Security Analysis of Cyber-Physical Systems Supporting SAE J3061," in *CPSData – Second International Workshop on modeling, analysis and control of complex Cyber-Physical Systems*, 2016, pp. 1–8.

[24] Kerry Maletsky, "RSA vs ECC Comparison for Embedded Systems," *White Paper, Atmel*, p. 5, 2015.

Paper D

M. Steger, C.A. Boano, M. Karner, J. Hillebrand, W. Rom, and K. Römer. **Generic Framework Enabling Secure and Efficient Automotive Wireless SW Updates.** *In Proceedings of the 21st International Conference on Emerging Technologies and Factory Automation (ETFA).*, pages 1–8. Berlin, Germany. November 2016.

Summary. In this paper a generic framework enabling wireless software updates as well as wireless diagnostics that is meant to support the whole life-cycle of a modern vehicle is proposed. The proposed framework considers the requirements coming from different application scenarios (i.e., vehicle development, vehicle assembly line, maintenance in workshops, and wireless remote updates and diagnostics). The paper highlights, that all addressed scenarios require fast, efficient, reliable, and secure wireless software updates. To enable efficient and fast wireless software updates, the proposed framework supports parallel software updates, where the same software binary will be installed on different vehicles and ECUs simultaneously. Additionally, the paper provides an overview on the defined security concept, explains the employed authentication and encryption mechanisms and describes how the integrity of the transferred data as well as of the entire vehicle is ensured. The proposed framework is finally evaluated by employing a developed vehicle and ECU model as well as by by connecting the implemented framework to a real vehicle to run wireless diagnostics.

My contributions. As main author of this paper I developed the wireless automotive software update framework and performed the related system evaluation. I wrote the vast majority of this paper in collaboration and discussion with the co-authors, who eminently supported the design and the description of the entire update framework as well as provided beneficial input regarding the system evaluation.

©2016 IEEE.
ISBN: 978-1-5090-1314-2.
DOI: 10.1109/ETFA.2016.7733575.

Link: http://ieeexplore.ieee.org/document/7733575/.

Generic Framework Enabling Secure and Efficient Automotive Wireless SW Updates

Marco Steger*, Michael Karner*, Joachim Hillebrand*, Werner Rom*, Carlo Boano†, and Kay Römer†

*Virtual Vehicle Research Center, Graz, Austria
†Institute for Technical Informatics, Graz University of Technology, Graz, Austria
Email: marco.steger@v2c2.at

***Abstract*—Future vehicles will be wirelessly connected to nearby vehicles, to the road infrastructure and to the Internet in order to enable new comfort features, safety functions and a number of new vehicle-specific services. The latter will include a fast, secure, and reliable way to remotely diagnose and reconfigure a vehicle as well as to install new software on the electronic control units integrated in a vehicle. Such wireless software updates are beneficial for both automotive OEMs and customers, as they allow to enable new features of the vehicle remotely and to fix software bugs by installing a new software version over the air. Wireless diagnostics and software updates are required in several stages of a vehicle's lifetime: from the manufacturing stage on the assembly line and the maintenance in a workshop to the remote download of up-to-date software directly by the car owner. To support this process over a whole vehicle's lifetime, a generic framework is needed. In this paper we propose a generic framework enabling secure and efficient wireless automotive SW updates and hence supporting a vehicle's whole lifetime. We describe the IEEE 802.11s network used as wireless medium to interconnect vehicles and diagnostic devices in a reliable, trustworthy and fast way and propose a dedicated cross-layer security concept applying strong authentication as well as encryption mechanisms.**

I. Introduction

A modern vehicle includes a growing number of electronic control units (ECU) allowing to incorporate new features and services. Enabling these features and services, however, requires elaborate and complex software (SW) implementations installed on these ECUs, potentially introducing a growing number of bugs in the automotive software.

New concepts allowing efficient automotive SW updates are required to fix such SW bugs, as well as to enable the remote installation of new features and to support the development and maintenance of modern vehicles. Efficient and secure SW updates can be beneficial over the whole life-cycle of a modern vehicle and can significantly reduce the time needed for vehicle maintenance.

In this paper we propose a generic framework enabling wireless SW updates as well as wireless diagnostics that is meant to support the whole life-cycle of a modern vehicle. Thereby, the proposed framework considers the requirements coming from different application scenarios (i.e., vehicle development, vehicle assembly line, maintenance in workshops, and wireless remote updates and diagnostics). In all these scenarios, the wireless SW updates have to be fast, efficient, reliable, and secure. To enable efficient and fast wireless SW updates, our framework will support parallel SW updates, where the same SW binary will be installed on different vehicles and ECUs simultaneously.

Additionally, we present a security concept that uses strong authentication and encryption mechanisms and also ensures the integrity of the transferred data as well as of the entire vehicle protecting the diagnostic devices, the transferred data, the vehicles and the OEM backbone from unauthorized access.

The vehicles and the diagnostic devices are interconnected using an IEEE 802.11s mesh network [1]. In [2] we already proved the applicability of IEEE 802.11s for automotive applications. The mesh characteristics of an IEEE 802.11s network increases the reliability as well as the flexibility of the network thanks to its multi-hop capability and the resulting redundant paths. The IEEE 802.11s standard also supports multicast data streams, which can be used to realize parallel and therefore efficient SW updates by sending a SW binary through the wireless network to several vehicles simultaneously.

After reviewing related work in the next section, we describe our generic framework and highlight the role of involved devices as well as users in the considered application scenarios in Section III. Thereafter, in Section IV, we illustrate the properties and advantages of the employed IEEE 802.11s [1] network, explain how it enables the construction of a large mesh network and illustrate how to exploit its characteristics to perform reliable and efficient wireless SW updates. Section V provides an overview of the performed security analysis as well as the resulting security concept. The results of the performed framework evaluation can be found in Section VI.

The key contribution of this paper is the introduction of a generic framework for wireless automotive SW updates that is applicable for different application scenarios, namely: wireless SW updates in the vehicle development, in the assembly line, during maintenance in workshops, as well as wireless remote updates. Our framework addresses the *efficiency* aspect by enabling parallel SW updates using IEEE 802.11s multicast data streams, and *security* by presenting a novel robust security concept for wireless SW updates.

978-1-5090-1314-2/16/$31.00 ©2016 IEEE

II. Related work

Wireless automotive SW updates were already addressed in previous work. In [3] the advantages of wireless SW updates compared to traditional wired SW updates are highlighted and a high-level architecture for over-the-air updates is presented. Technical details such as a description of the wireless medium or required security mechanisms, however, are not covered.

Idrees et al. [4] proposed a system for wireless over-the-air updates, where a HW Security Module (HSM) is used for data encryption, data integrity, as well as key management on both the wireless interface and all ECUs of a vehicle. Data encryption and verification is handled directly on the concerned ECU. This approach, however, requires an HSM on every ECU, which leads to significant extra costs. Additionally, the proposed system does not describe the specific type and properties of the wireless link.

Nilsson et al. [5], [6] have proposed a system, where a vehicle is connected to a portal server via an Internet link to install new automotive SW over the air. The authors highlight the importance of the security aspects data integrity and data confidentiality for wireless SW updates. However, neither the wireless network nor the used wireless protocol and the data flow in the network have been addressed.

Mahmud et al. [7] have proposed an architecture for secure SW updates in which the integrity of data is ensured by sending multiple copies of the SW. The proposed solution, however, is dedicated to unicast links between one vehicle and an OEM sever and relies on a number of prerequisites and assumptions (e.g., ensuring secure SW updates by sending multiple copies).

Although the authors are addressing several security aspects, the solutions described above are only applicable for wireless point-to-point links and only the remote SW update scenario is covered. Multicast data streams and hence parallel SW updates cannot be realized.

Tesla is currently the only OEM providing a solution for wireless remote updates over the air. To this end, Tesla is using a dedicated wireless link (mainly 3G, but a vehicle can also be connected to the user's Wi-Fi at home) to securely transfer new SW from the manufacturer servers to a vehicle [8]. This point-to-point connection (i.e., between server and vehicle), however, cannot be used to simultaneously install SW in different vehicles and on several ECUs in parallel.

The authors of [4]–[7] have specifically focused on secure wireless SW updates and highlighted a number of security threats and aspects, namely:

- Vehicle integrity and authentication
- Data integrity
- Data confidentiality
- Key management and exchange

Different from previous work, our security concept presented in Section V-A addresses all these aspects at once.

A. IEEE 802.11s Mesh Network

Our generic framework for wireless SW updates is based on an IEEE 802.11s mesh network. The IEEE 802.11s standard provides multicast data streams and therefore we can realize parallel SW updates in our framework.

To the best of our knowledge there are no other wireless SW update solutions based on IEEE 802.11s available, however, IEEE 802.11s is used in other automotive applications.

For example, in [9] and [10], IEEE 802.11s is used as a backbone network for V2X (vehicle to vehicle and vehicle to infrastructure communication) networks. The main idea is to replace the wired connections between the RSUs (road side units) and the V2X servers by wireless ones. In [2] we proved the applicability of IEEE 802.11s for an automotive application by performing different experiments in an automotive environment. These experiments, however, were not dedicated to wireless SW updates but focusing on reliable wireless data transfer in an automotive context.

In Section IV we describe the mesh characteristics of the used IEEE 802.11s network in more detail and highlight the advantages of the latter w.r.t. wireless automotive SW updates.

B. IEEE 802.11s Security

Several contributions have focused on the security aspects of IEEE 802.11s and proposed possible improvements and extensions. These aspects, however, have only been discussed outside of the automotive domain. Tan et al. [11] have described internal as well as external attacks on IEEE 802.11s networks. Security requirements for systems based on IEEE 802.11s were extracted and these requirements were used to evaluate different approaches. The results of this evaluation have shown that none of the evaluated approaches (e.g., [12] and [13]) was able to satisfy the desired security requirements.

Sbeiti et al. [14] have proposed to use GPS positioning to mitigate a wide range of potential attacks. The system uses a combination of digital signatures, lightweight authentication trees and symmetric block ciphers called PASER. In [15] the same authors also evaluate and compare other approaches for secure IEEE 802.11s networks to PASER. They highlight the inefficiency of these approaches w.r.t. time (delay of about 70ms per-hop) and power as well as their sensitivity to blackhole and wormhole attacks. However, the use of GPS, as proposed in [14] and [15], is not feasible inside a workshop building or the assembly line and therefore PASER is not applicable for our wireless SW update system.

Although several approaches to create secure IEEE 802.11s networks exist [11]–[15], none of these approaches is fulfilling both the efficiency requirements of our framework for wireless SW updates and the immunity against possible attacks. Because of that, our security concept described in Section V is using a lightweight security mechanism based on pre-shared keys on the network layer and additional strong authentication as well as encryption mechanisms on the application layer.

In particular, our security concept is based on a structured security analysis. In [16] we have described the security analysis in detail and showed how it can be used to analyze an automotive application w.r.t. the new automotive security standard SAE J3016 [17]. A security concept enabling wireless SW updates, however, was not part of this work.

III. Framework Description and System Overview

In this section we first describe the considered application scenarios for wireless SW updates and diagnostics in more detail (Section III-A). Thereafter, in Section III-B, we list the involved nodes, devices, and tools and sketch the data flow between these entities in our framework.

A. Considered Application Scenarios

In this section we use the following descriptions of the considered application scenarios to identify the scenario-specific requirements as well as its peculiarities and describe the corresponding environment w.r.t. the user education, available infrastructure, and security concerns. In Table I these aspects are summarized.

In the **vehicle development scenario**, development engineers have to update the SW of one or more ECUs several times to evaluate and test newly developed features. A flexible and efficient framework for SW updates and vehicle diagnostics is required to support the development engineers in their work. Vehicle development activities will primarily take place in a restricted environment and performed by expert users.

Vehicle assembly is performed in a highly automated and secure environment where many working steps are performed by machines and robots. Before a vehicle can leave the assembly line, the latest SW version shall be installed on all integrated ECUs. Therefore, the SW of many vehicles must be updated – ideally in parallel – to install the latest SW on the ECUs of these vehicles. Because of the high number of vehicles as well as the high degree of automation, the scalability, reliability and efficiency of the SW update system are very important.

In a typical **workshop scenario** mechanics will diagnose, repair and maintain several vehicles. Therefore, the mechanics (trained users) will connect to a vehicle, run some diagnostic functions, look for Diagnostic Trouble Codes (DTC) on the ECUs and perform necessary repairs. If new SW is available for a vehicle, the mechanics additionally install the new SW. Parallel SW updates would be very beneficial especially when large vehicle recalls (e.g., due to a critical SW bug) are necessary: a mechanic can connect to several vehicles in parallel and install the new SW simultaneously.

Remote SW updates are mainly relevant for future vehicles with an integrated wireless interface to the vehicle. To remotely install new SW on a vehicle, the vehicle owner will first be informed about the possible update. In the next step, the user will choose a suitable time slot for the SW update (the vehicle cannot be used while a SW update is performed due to safety reasons). The data will then be transferred to the vehicle either via a dedicated 3G/4G connection or when the vehicle owner connects the vehicle to his home WLAN.

In Table I we collected the application-scenario-specific information w.r.t. wireless SW updates. Thereby we focus on the user and his education, important requirements w.r.t. reliability, efficiency, etc., as well as the required security level.

SW updates in the assembly line as well as in the vehicle development phase are performed in secured areas by expert users. Therefore, security is still an important issue (e.g., industrial espionage) but not as critical as in the remote SW update case, were a SW update is performed in public or at the users home, potentially using an insecure network by an untrained user. Especially the assembly line as well as the workshop scenario require a very efficient and fast way to install SW updates. High flexibility by easily extending the transmission range of the wireless network is especially required during vehicle development due to the big variety of system evaluations, function tests and diagnostics performed in this phase.

TABLE I
INVOLVED USERS, ASPECTS AND REQUIRED SECURITY LEVEL W.R.T. WIRELESS SW UPDATES IN THE CONSIDERED APPLICATION SCENARIOS.

Scenario	User	Aspects	Security Level
Development	Development engineer; expert	Flexible, efficient	Medium
Assembly line	Operator; expert	Scalable, reliable, efficient	Medium
Workshop	Mechanic; trained user	Efficient, backward compatibility	High
Remote	Vehicle driver or owner; untrained user	Easy to use	Very high

B. System Overview

To interconnect the vehicle with the wireless network, a reliable and secure interface is required. This **Wireless Vehicle Interface (WVI)** is the core component of our framework for wireless SW updates. Independent from the application scenario, the WVI interconnects the in-vehicle communication system, consisting of different automotive bus systems (e.g., CAN) and ECUs, with the wireless network and therefore with the outside world.

We see two different types of WVIs: either a fully integrated device or a plug-in solution. The plug-in WVI can be temporarily connected to the vehicle via the OBD interface. This is mainly relevant for the workshop scenario, where the plug-in property and the use of standardized protocols for the in-vehicle communication are ensuring backward compatibility. The fully integrated WVI is either realized as a dedicated ECU or as a component of a smart gateway interconnecting different bus systems and therefore part of the in-vehicle communication system.

The **Diagnostic Tester (DT)** is either a dedicated device (e.g., tool in a workshop) or more likely a SW application running on a laptop, PC or server. The DT provides information about the vehicle configuration (e.g., type and CAN IDs of the ECUs), availability of new SW versions, as well as keys to authorize a SW update. The information is either stored locally on the device or can be acquired using a dedicated and trusted connection to an OEM server. Additionally, a DT supports various diagnostic functions. In the remote SW update scenario the DT will be located at a facility of the OEM and a secure Internet link (e.g., a VPN tunnel) between the WVI and the DT will be established.

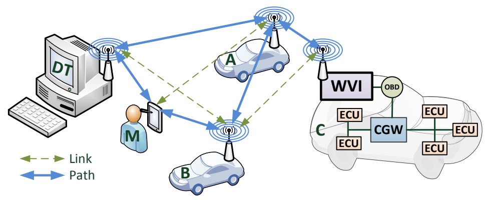


Fig. 1. IEEE 802.11s wireless mesh network applied in a typical workshop scenario. A mechanic M can connect to a vehicle either directly (e.g., vehicle B) or using a multi-hop path (e.g., vehicle C via vehicle B and vehicle A).

Handheld devices can be used to trigger, monitor, and validate the SW update process. Additionally, they can be used by mechanics in a workshop or by a development engineer in the vehicle development phase to run wireless diagnostics and to monitor the vehicle bus.

C. Setup Phase and Data Flow

In Figure 1 a typical workshop scenario is sketched, where mechanics maintain vehicles by using handhelds to run wireless diagnostics and trigger wireless SW updates. We will use the illustrated scenario to explain the setup phase and the data flow in our framework: similar procedures are used in the other scenarios. To simplify the description of the processes below, we will not cover the performed security steps. However, in Section V-A we will refer to this example again to illustrate our security concept.

A mechanic first connects a WVI to the vehicle to be maintained using the OBD interface. The WVI powers up and joins the wireless IEEE 802.11s network. The WVI now waits for beacons advertising the presence of a DT. These beacons are broadcasted periodically and once the WVI receives those beacons, it will establish a connection to the DT. Also handhelds can connect to the WVI by sending beacons. The mechanic can now use the connected handheld device to run wireless diagnostic and, if a new SW version is available, will trigger a wireless SW update. Therefore, the SW will be transferred from the DT to the WVI. Note that a new SW version is not stored on the handheld devices due to security reasons.

Our generic framework supports different SW update modes: most likely the SW binary will first be fully transferred from the DT to the WVI. The latter will then verify the received binary, and install the binary on the ECU using the Unified Diagnostic Service (UDS) protocol [18] (*two-stage approach*). When using the *direct-programming mode*, the DT will send packets containing UDS-complaint data chunks to the WVI and the latter unpacks the data and forwards it to the ECU. Therefore, the WVI only acts as a gateway forwarding the data and the DT has to take care about authorization with the ECU, data transfer, and verification of the SW update. Our framework can also support *delta downloads*, where only the difference between the current and the new SW version is sent to the ECU. This mode, however, requires that also the

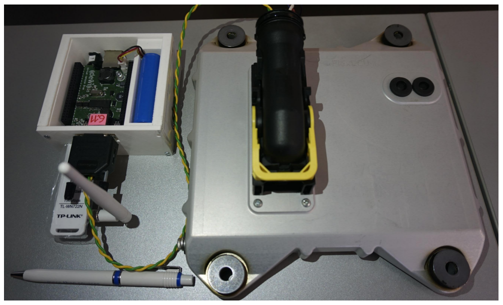


Fig. 2. WVI prototype based on a BeagleBone Black (left) and the Volvo ECU used to test the SW update (right).

ECU itself supports this delta download mode (which is not yet fulfilled by current ECUs).

To perform a SW update on several vehicles in parallel, a mechanic will register a vehicle for a certain SW update and the DT will then automatically start the update process when all vehicles are registered and ready to receive the new SW.

D. Prototype Implementation

We developed prototypes of WVI, DT, and handheld device to evaluate the framework, the data flow, and the individual components.

The WVI prototype is shown in Figure 2 and consists of a BeagleBone Black (BBB) board and an additional, self-designed PCB, the DEWI[1] cape. This cape includes the required HW interfaces (i.e., CAN and OBD) and the corresponding transceivers to connect the WVI with a vehicle. Additionally, the cape is handling the battery management of the WVI and provides some means to measure the power consumption of the board as well as of the individual radio. The WVI SW is mainly developed in Java and the Java Native Interface (JNI) is used to control the HW-related parts.

The DT is implemented in Java and the implementation was tested on a dual-core laptop running Win7 as well as on a PC running Debian Linux. We developed a GUI providing different modes such as ECU programming, OBD diagnostics, and CAN bus monitoring.

A Nexus 7 tablet running Android is used as handheld device. The current version of the developed Android application allows to connect to a WVI and a DT simultaneously, to monitor the vehicle bus, and to exchange status information with the DT. Currently, we are extending the application to fully support the defined security mechanisms. The new version of the application can then be used to trigger, monitor and validate the SW update process.

[1]The ARTEMIS project DEWI (Dependable Embedded Wireless Infrastructure) focuses on the area of wireless sensor / actuator networks and wireless communication. With its four industrial domains (Aeronautics, Automotive, Rail, and Building) and 21 industry-driven use cases, DEWI will provide and demonstrate solutions for wireless seamless connectivity and interoperability in smart infrastructures [19]. (For further details see www.dewi-project.eu)

IV. Wireless IEEE 802.11s Network

Our generic framework supporting efficient and secure wireless SW updates as well as wireless diagnostics uses an IEEE 802.11s network to interconnect the devices described in Section III. Contrary to other standards out of the IEEE 802.11 protocol family, where an access point is used to interconnect the nodes of a network, the IEEE 802.11s protocol is based on a mesh network, and each node can directly communicate with other nodes in its transmission range or use other nodes in between to forward a data packet to its final destination [20]. Because of the mesh characteristics of IEEE 802.11s, a data packet can use different paths when sent through the network and this redundancy increases the reliability of IEEE 802.11s networks.

Additionally, the transmission range of the network can be increased by adding relay nodes at the edge of the network. Thanks to the multihop capability of mesh networks, an IEEE 802.11s network can even be used in difficult environments (for wireless communications) and thus can fulfill the requirements of the considered application scenarios.

For example, in a typical workshop scenario, wireless links will be affected by the shielding of the vehicles and other (metal) objects: if the direct link between a DT and a vehicle is too weak to send a packet directly from the DT to the WVI, other vehicles parked in between may be used to forward the data packet to the target vehicle (also illustrated in Figure 1).

A. Links and Paths in IEEE 802.11s

Figure 1 illustrates a mesh network with five nodes in a typical workshop scenario. The Hybrid Wireless Mesh Protocol (HWMP), used as the default routing algorithm in IEEE 802.11s networks, automatically finds the best (reliable and fast) route between all nodes in the network [20]. In an IEEE 802.11s network we have to distinguish between links and paths. An established link between two nodes (e.g., vehicles B and C) means that the nodes are in transmission range of each other. However, this doesn't always mean that data between these nodes is exchanged directly. If a link is weak (e.g., several packets are lost when sent directly from vehicle B to vehicle C) the HWMP tries to find a better way to route the packets through the network. In IEEE 802.11s these routes are called paths (e.g., between vehicle A and B or mechanic M and DT).

B. Multicast in IEEE 802.11s

In our framework we use open11s [21], an open source implementation of the IEEE 802.11s standard available for Linux. open11s is still under development and doesn't include all features described in the IEEE 802.11s standard. However, as open11s is open-source, missing features can be added using (self-implemented) patches.

The IEEE 802.11s standard also describes multicast data streams in the mesh network. Such a multicast can be used to send data packets from one node to several other nodes in a network. In our framework we use multicasts to transfer and install a new SW version on several vehicles simultaneously (i.e., parallel SW updates). Although the IEEE 802.11s standard defines multicast data streams, the current open11s implementation does not support multicasts yet. We are currently evaluating the reliability of experimental patches enabling multicast data exchange for open11s in our IEEE 802.11s testbed consisting of twelve nodes. In parallel we plan to develop our own reliable multicast and to compare the performance of the different approaches using our wireless testbed.

The framework for wireless SW updates and diagnostics as well as the corresponding security concept presented in Section V, however, is already designed and implemented in a way that it will fully support parallel SW updates.

V. Security and Trust

Security is a critical aspect of wireless SW updates. Depending on the application scenario, different levels of security are required (as described in Section III-A). In this section we briefly describe how security is addressed in our framework and illustrate the required steps towards a security concept for wireless SW updates. These steps can be summarized as follows:

1) ***Security analysis*** of the framework resulting in a secure system configuration.
2) ***Extraction of the security requirements*** from the secure system configuration.
3) ***Definition of a security concept*** based on the security requirements and the peculiarities of the application scenario.
4) ***Evaluation of the defined security concept*** using the STRIDE threat model [22].

We used the DEWI security metric [16] to perform a structured security analysis. The DEWI security metric is based on the SHIELD multi-metrics approach [23], [24] and can be used to analyze the different application scenarios described in Section III at once. The analysis results in a secure system configuration for every application scenario and the security-related requirements can be directly extracted from these secure system configurations. In [16] we described the DEWI security metric in more detail and presented a case study illustrating its usage.

A. Security Concept

The security concept is based on the results of the structured security analysis using the DEWI security metric: the secure system configurations and the corresponding security requirements.

As described in Section II, our security concept applies security mechanisms on the network as well as on the application layer. We utilize wpa_supplicant, a generic security framework for different types of wireless networks, to secure the wireless IEEE 802.11s network in a lightweight way based on existing mechanisms provided by the current open11s implementation. SAE [25], developed especially for 802.11s-based, multi-hop capable mesh networks, is fully integrated in the latest wpa_supplicant version and thus can be used to secure our wireless IEEE 802.11s mesh network.

On the application layer different keys and security mechanisms are used. Thereby our security concept addresses the four security aspects stated in Section II: vehicle integrity and authentication, data integrity, data confidentiality, key management and exchange.

Vehicle integrity and authentication require strong authentication mechanisms. Additionally we have to avoid that an attacker can endanger a whole fleet of vehicles by breaking one vehicle and extracting the shared authentication key. Thus, a unique key shall be used on every vehicle. In our concept we are using a key pair consisting of a private and a public key (asymmetric keys) on the WVI as well as the DT ensuring an unambiguous authentication between all nodes without using the same (symmetric) key on all vehicles. This *master key pair* is used to handle the **authentication between a WVI and a DT** and to **encrypt as well as sign unicast data**.

Secure multicasts needed to enable parallel SW updates, however, require symmetric keys for data encryption and verification. A data packet sent from the DT to several vehicles must be encrypted using a shared key. This shared *session key* is created by the DT and then sent from the DT to each WVI using a unicast data packet encrypted with the public master key of the WVI and signed with the private master key of the DT. The session key can then be used to encrypt and sign the multicast data packets and hence ensuring **data confidentiality** as well as **data integrity**.

All keys used in our security concept must be kept secret and therefore stored securely on the devices to ensure that an attacker cannot extract the keys (e.g., by stealing a WVI). In our concept, keys are either stored in dedicated secure memory or a trusted platform module is used to securely hold the keys.

We illustrate a typical workshop scenario in Section III-B and Figure 1. We now refer to the same scenario and discuss the related steps w.r.t. security:

1) ***User authentication***: the mechanic authenticates with the system using a smartcard and a PIN code. Different user profiles can be used to authorize different features: a normal mechanic can use the system to run wireless diagnostics only. A privileged user can additionally perform wireless SW updates.
2) ***WVI power up***: after connecting the WVI to the vehicle, the WVI powers up and tries to connect to the IEEE 802.11s network. Therefore, a shared network key is used.
3) ***Authentication with the DT***: the master keys of the DT and the WVI are used to authenticate with each other.
4) ***Parallel SW update***: to enable parallel SW updates, the DT first creates a session key and sends it to every concerned vehicle's WVI (unicast). Now the DT can send the SW binary to the WVIs. Thereby every transferred packet is encrypted and signed using the session key (multicast).
5) ***Data verification***: after transferring the whole binary to the WVIs, the DT computes the hash value of the entire binary, signs and encrypts the hash value using the master key and sends it to the WVIs (unicast). The WVIs can verify the received SW binary using the received hash before installing it on the concerned ECUs.

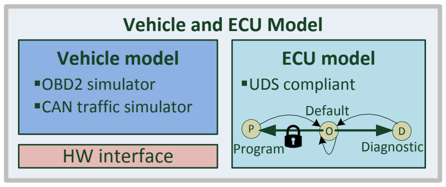


Fig. 3. Vehicle and ECU model consisting of a vehicle model, simulating traffic on a CAN bus and supporting some OBD requests, and an ECU model, implemented as state machine and supporting UDS (required for SW updates)

A more detailed description as well as a STRIDE based evaluation of our security concept can be found in [26].

VI. Framework Evaluation

In this section, we present an evaluation of our framework for secure and efficient wireless SW updates and diagnostics. The evaluation is performed in three steps: first we use our developed Vehicle and ECU Model (VEM) to perform fundamental experiments and communication tests, as well as to test the behavior in case of errors (Section VI-B). In the second step, we wirelessly connect to a real vehicle and run wireless diagnostics (Section VI-C). Finally, we evaluate our framework by installing a new SW binary on an automotive ECU provided by our project partners Volvo Trucks (Section VI-D).

A. Vehicle and ECU Model

We use the VEM to perform fundamental system evaluations, analysis of new features, and communication tests. Furthermore, the VEM can also be used to evaluate the behavior of the developed system in case of errors (e.g., sending unexpected frames) and communication problems (e.g., lost, delayed or duplicated CAN frames).

As illustrated in Figure 3, the developed VEM consists of a fundamental communication model of a vehicle as well as a simplified model of an ECU and is implemented in C/C++. A developed HW interface library can be used to communicate with a vehicle via the OBD or CAN interface.

The vehicle model can be used (i) to simulate the traffic on a CAN bus and (ii) to create a response for certain OBD requests. Bus traffic simulation is done by using log files taken from different vehicles by connecting our WVI to the OBD interface of the vehicles to collect the timestamps, CAN-IDs, and payloads of all received CAN frames. The vehicle model parses such a log file and writes the collected messages on the CAN accordingly.

The ECU model is implemented as state machine covering different diagnostic sessions described in the UDS standard [18]. This state machine is also sketched in Figure 3. By default, the ECU will power up and stay in operational mode,

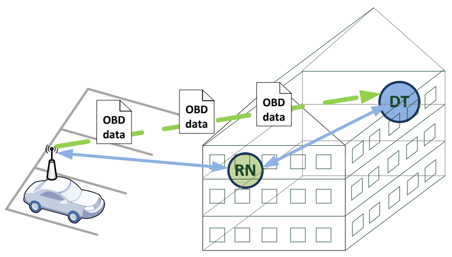


Fig. 4. Evaluation of the wireless diagnostic feature: connecting a real vehicle with the DT located in the second floor via a relay node (RN) located in the first floor.

where the ECU is performing its dedicated task. A UDS *diagnostic session control* command can be used to switch the ECU in the *programming session* or in the *extended diagnostic session* [18]. Depending on the state of the ECU (i.e., on the current UDS session) different diagnostic commands are available. An authorization step, often based on a seed & key mechanism, must be performed to empower the programming session. If the ECU is in the extended diagnostic or programming session and the session is inactive (i.e., no UDS messages received for a predefined time), the ECU falls back to the default session.

Our current ECU model supports all programming-related commands, as well as the *diagnostic session control*, *security access*, *ECU reset*, and *tester present* commands as defined in the UDS standard [18].

B. Framework development and evaluations using the VEM

The first experiments and evaluations were performed using the developed VEM. Therefore, the VEM is connected with the WVI using (i) a virtual CAN bus, where the VEM and the WVI implementation is running on the same BBB board or (ii) using a CAN connection between the WVI prototype and the VEM, running on a second BBB.

In both cases (i.e., WVI and VEM on the same BBB or VEM and WVI on their own BBB) we can test the wireless diagnostic as well as the ECU programming part of the developed system using the VEM. Therefore, the VEM can be used to continuously evaluate and test the currently developed versions of the DT and WVI prototypes.

C. Wireless diagnostics in a real vehicle

We also evaluate our framework by running wireless diagnostics on a real vehicle using our WVI prototype. As illustrated in Figure 4, the DT, running on a PC located on the second floor of our company building, is connected to the WVI via another WVI prototype, located on the first floor and acting as Relay Node (RN).

Using this experimental setup, we are able to monitor the CAN bus that is available at the OBD connector of the vehicle and use different OBD requests to read vehicle-specific parameters such as the current engine temperature or the current engine speed (RPM). However, some particular OBD requests/responses were heavily delayed or lost during transmission. We performed additional measurements using the described scenario to get accurate numbers for the Packet Error Rate (PER). Therefore we sent 10000 UDP packets with 8 bytes payload and a period of 100ms from the WVI, connected to the vehicle, to the DT, located in the office. An acknowledgment packet with the same size is then sent back to the WVI. This experiment was performed without any retransmission mechanisms on application layer and results in a PER of 0.43% for messages sent from the WVI to the DT. 0.83% of all packets sent back from the DT to the WVI were also lost. In total we lost 0.65% of all transferred packets.

Further investigations using our IEEE 802.11s testbed are planned to evaluate and increase the reliability of the network.

D. Wireless SW update using a real ECU

The SW update process was evaluated using a demo ECU provided by our project partner Volvo T rucks. Due to organizational and especially safety reasons, a SW update test in a real vehicle is not possible yet in the current stage of the project.

As shown in Figure 2, the WVI is directly connected to the demo ECU using a twisted-pair cable. The demo ECU is UDS-compliant and can be programmed in two steps: first a secondary bootloader is installed and executed. Afterwards the application binary can be installed. A reboot of the ECU is used to complete the SW update process.

Volvo Trucks provided different SW versions periodically writing CAN frames on the bus when in operational (default) mode. Every version uses its own CAN-ID and therefore we can easily verify if a new version is installed on the demo ECU or not.

The first version of the WVI HW causes some problems when connecting to the Volvo demo ECU using the CAN bus. Seconds after connecting our WVI to the demo ECU, the latter got caught in a reset loop due to a bad CAN bus state. After investigating and fixing the problem by replacing the CAN transceiver chips, we are now able to connect our WVI (revision 2) to the demo ECU and install new SW versions on it without any problems.

The SW update process for the Volvo ECU is done in two steps. First the Secondary Bootloader (SBL) is transferred to the ECU and then a UDS command is used to launch this SBL on the latter. In the second step, the actual application binary is transferred and installed on the ECU. Our framework support both the use of an SBL and the application binary as well as an approach, where the application binary is directly installed without using an SBL.

In Table II the times needed to transfer the SBL and the application from the DT to the WVI and then from the WVI to the ECU are shown. The table shows that the wireless data transfer (including data transfer, integrity check, and acknowledgments) is 12 times faster then the wired one using the CAN bus. Thus, it make sense that the WVI autonomously control the data transfer to the ECU once the binary was received from the DT (i.e., no permanent wireless connection

TABLE II
TIME NEEDED TO TRANSFER THE BINARIES FROM THE DT TO THE WVI VIA THE IEEE 802.11S LINK AND THEN SEND IT TO THE ECU VIA CAN

Binary type	Binary size	Time on 11s link	Time on CAN
SBL	67KB	0.503s	6.268s
Application	375 KB	2.527s	30.664s

to the DT needed). The WVI then informs the DT when the SW is installed on the ECU successfully or any problems occur.

VII. CONCLUSION

In this paper we proposed a generic framework enabling secure and efficient wireless SW updates for vehicles. The framework is designed such that it can fulfill the needs of several application scenarios: vehicle development, vehicle assembly line, vehicle maintenance, and wireless remote updates. The proposed framework enables parallel SW updates, where the SW of several vehicles is updated simultaneously. We also describe the properties and characteristics of the IEEE 802.11s standard and its applicability for performing wireless SW updates. Furthermore, we describe our approach to enable secure automotive SW updates and especially focus on the resulting security concept.

We are currently working on the extension and improvement of the open11s implementation with specific focus on reliable and secure multicasts and the stability of the paths in the network. The revised version of open11s will then be used to evaluate the performance of our system w.r.t. parallel SW updates.

ACKNOWLEDGMENT

We are particularly grateful to our colleagues from Volvo Trucks for their great support regarding the demo ECU.

The research from DEWI project (www.dewi-project.eu) leading to these results has received funding from the ARTEMIS Joint Undertaking under grant agreement n° 621353. The authors acknowledge the financial support of the COMET K2 - Competence Centres for Excellent Technologies Programme of the Austrian Federal Ministry for Transport, Innovation and Technology (BMVIT), the Austrian Federal Ministry of Science, Research and Economy (BMWFW), the Austrian Research Promotion Agency (FFG), the Province of Styria and the Styrian Business Promotion Agency (SFG).

REFERENCES

[1] IEEE, “Local and metropolitan area networks-Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY):Amendment 10: Mesh Networking,” IEEE, Tech. Rep., 2011.

[2] M. Steger, M. Karner, J. Hillebrand, W. Rom, E. Armengaud, M. Hansson, C. A. Boano, and K. Romer, “Applicability of ieee 802.11s for automotive wireless software updates,” in *Telecommunications (ConTEL), 2015 13th International Conference on*, July 2015, pp. 1–8.

[3] Redbend Software, “Updating Car ECUs Over-The-Air (FOTA),” *White Paper*, pp. 1–14, 2011.

[4] M. S. Idrees, H. Schweppe, Y. Roudier, M. Wolf, D. Scheuermann, and O. Henniger, “Secure automotive on-board protocols: A case of over-the-air firmware updates,” *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 6596 LNCS, pp. 224–238, 2011.

[5] D. K. Nilsson and U. E. Larson, “Secure firmware updates over the air in intelligent vehicles,” *IEEE International Conference on Communications*, pp. 380–384, 2008.

[6] D. Nilsson, P. Phung, and U. E. Larson, “Vehicle ECU classification based on safety-security characteristics,” *Road Transport Information and Control - RTIC and ITS United Kingdom Members’ Conference, IET*, pp. 1–7, 2008.

[7] S. M. Mahmud, S. Shanker, and I. Hossain, “Secure software upload in an intelligent vehicle via wireless communication links,” in *Intelligent Vehicles Symposium. Proceedings. IEEE*, June 2005, pp. 588–593.

[8] N. Gabe, “Over-the-air updates on varied paths,” *Automotive News*, 2016-01-25.

[9] D. T. Tuan, S. Sakata, and N. Komuro, “Priority and admission control for assuring quality of I2V emergency services in VANETs integrated with Wireless LAN Mesh Networks,” *ICCE 2012*, pp. 91–96, 2012.

[10] S. Chakraborty and S. Nandi, “IEEE 802.11s mesh backbone for vehicular communication: Fairness and throughput,” *IEEE Transactions on Vehicular Technology*, vol. 62, no. 5, pp. 2193–2203, 2013.

[11] W. K. Tan, S.-G. Lee, J. H. Lam, and S.-M. Yoo, “A security analysis of the 802.11s wireless mesh network routing protocol and its secure routing protocols,” *Sensors*, vol. 13, no. 9, p. 11553, 2013.

[12] M. S. Islam, M. A. Hamid, and C. S. Hong, “SHWMP: A Secure Hybrid Wireless Mesh Protocol for IEEE 802.11s Wireless Mesh Networks,” in *Transactions on Computational Science VI*. Springer Berlin Heidelberg, 2009, pp. 95–114.

[13] J. Ben-Othman and Y. I. Saavedra Benitez, “IBC-HWMP: a novel secure identity-based cryptography-based scheme for Hybrid Wireless Mesh Protocol for IEEE 802.11s,” *Concurrency and Computation: Practice and Experience*, vol. 25, no. 5, pp. 686–700, 2013.

[14] M. Sbeiti, A. Wolff, C. Wietfeld, and I. Technology, “PASER: Position Aware Secure and Efficient Route Discovery Protocol for Wireless Mesh Networks,” in *International Conference on Emerging Security Information, Systems and Technologies - SECURWARE*, 2011.

[15] M. Sbeiti and C. Wietfeld, “One stone two birds: On the security and routing in wireless mesh networks,” in *Wireless Communications and Networking Conference (WCNC), 2014 IEEE*, April 2014.

[16] M. Steger, M. Karner, J. Hillebrand, W. Rom, and K. Romer, “A Security Metric for Structured Security Analysis of Cyber-Physical Systems Supporting SAE J3061,” in *CPSData – Second International Workshop on modeling, analysis and control of complex Cyber-Physical Systems*, 2016, pp. 1–8.

[17] SAE, “SAE J3061: Surface Vehicle Recommended Practive - Cybersecurity Guidebook for Cyber-Physical Vehicle Systems,” SAE International, Tech. Rep., 2016.

[18] ISO, “Road vehicles Unified diagnostic services (UDS) Specification and requirements,” ISO 2006, Tech. Rep., 2006.

[19] W. Rom, P. Priller, J. Koivusaari, M. Komi, R. Robles, L. Dominguez, J. Rivilla, and W. Van Driel, “DEWI – Wirelessly into the Future,” in *2015 Euromicro Conference on Digital System Design (DSD)*, Aug 2015, pp. 730–739.

[20] G. R. Hiertz, D. Denteneer, S. Max, R. Taori, J. Cardona, L. Berlemann, and B. Walke, “IEEE 802.11s: The WLAN mesh standard,” *IEEE Wireless Communications*, pp. 154–160, 2010.

[21] “open80211s – An open-source implementation of the recently ratified IEEE 802.11s wireless mesh standard,” http://open80211s.org/open80211s/.

[22] B. Potter, “Microsoft SDL threat modelling tool,” *Network Security*, pp. 15–18, 2009.

[23] I. Garitano, S. Fayyad, and J. Noll, “Multi-metrics approach for security, privacy and dependability in embedded systems,” *Wirel. Pers. Commun.*, vol. 81, no. 4, pp. 1359–1376, Apr. 2015.

[24] J. Noll, I. Garitano, S. Fayyad, E. Asberg, and H. Abie&, “Measurable security, privacy and dependability in smart grids,” *Journal of Cyber Security*, vol. 3, pp. 371–398, 2015.

[25] D. Harkins, “Simultaneous authentication of equals: A secure, password-based key exchange for mesh networks,” in *Second International Conference on Sensor Technologies and Applications. SENSORCOMM ’08.*, Aug 2008, pp. 839–844.

[26] M. Steger, C. Boano, M. Karner, J. Hillebrand, W. Rom, and K. Romer, “SecUp: Secure and Efficient Wireless Software Updates for Vehicles,” in *Under Submission*, 2016, pp. 1–8.

Paper E

M. Steger, A. Dorri, S.S. Kanhere, K. Römer, R. Jurdak, and M. Karner. **Secure Wireless Automotive Software Updates using Blockchains – A Proof of Concept.** *In Proceedings of the 21st International Forum on Advanced Microsystems for Automotive Applications (AMAA).*, pages 137–149. Berlin, Germany. September 2017.

Summary. In this paper an automotive security architecture utilizing Blockchain is proposed. The proposed architecture is able to tackle the implicated security and privacy challenges of future connected vehicles. The Blockchain-based security architecture can be utilized to perform over-the-air updates for smart vehicles remotely as well as to securely distribute the latest software to service centers or vehicle assembly lines where the latest software image is installed on the ECU of a vehicle locally. The proposed architecture ensures a secure as well as tamper-proof data exchange and protect the privacy of the end user. Thus, the proposed security architecture is not only applicable for protecting wireless automotive software updates but can also be utilized in a more general manner to secure a wide range of (future) automotive services. The proposed architecture is evaluated using a proof-of-concept implementation of a wireless software update system providing a secure as well as efficient communication between all involved parties. This implementation is used to i) show the applicability of a Blockchain-based architecture for wireless automotive software updates, ii) analyze the packet overhead of the architecture due to the use of Blockchain, iii) highlight its advantages compared to centralized (e.g., certificate-based architecture), and iv) evaluate the added latency compared to locally performed wireless software updates.

My contributions. I am one of the main authors of this paper and wrote several sections in collaboration and discussion with the co-authors. In particular, I developed the proof-of-concept implementation of the Blockchain architecture as well as a certificate-based architecture used as baseline system, and carried out the architecture evaluation (including a comparison with the baseline system). Ali Dorri significantly contributed to this paper by providing all his expertise about the Blockchain and played a vital role when defining the blockchain-based security architecture. Furthermore, he supported the development of the proof-of-concept implementation by providing input w.r.t. to Blockchain internals.

©2017 Springer International Publishing.
ISBN: 978-3-319-66972-4.
DOI: 10.1007/978-3-319-66972-4_12.

Link: https://doi.org/10.1007/978-3-319-66972-4_12.

Secure Wireless Automotive Software Updates Using Blockchains: A Proof of Concept

Marco Steger, Ali Dorri, Salil S. Kanhere, Kay Römer, Raja Jurdak and Michael Karner

Abstract Future smart vehicles will employ automotive over-the-air updates to update the soft ware in the embedded electronic control units. The update process can affect the safety of the involved users, thus requires a comprehensive and elaborate security architecture ensuring the confidentiality and the integrity of the exchanged data, as well as protecting the privacy of the involved users. In this paper, we propose an automotive security architecture employing Blockchain to tackle the implicated security and privacy challenges. We describe our proof-of-concept implementation of a Blockchain-based software update system, use it to show the applicability of our architecture for automotive systems, and evaluate different aspects of our architecture.

Keywords Automotive security architecture · Blockchains · Wireless software update · Over-the-air updates · Security · Privacy · Scalability

M. Steger (✉) · M. Karner
Virtual Vehicle Research Center, Inffeldgasse 21/a, 8010 Graz, Austria
e-mail: marco.steger@v2c2.at

M. Karner
e-mail: michael.karner@v2c2.at

A. Dorri · S.S. Kanhere
School of Computer Science and Engineering (CSE), University of New South Wales (UNSW), Sydney, Australia
e-mail: alidorri.ce@gmail.com

S.S. Kanhere
e-mail: salil.kanhere@unsw.edu.au

K. Römer
Institute for Technical Informatics, Graz University of Technology, Graz, Austria
e-mail: roemer@tugraz.at

R. Jurdak
Commonwealth Scientific and Industrial Research Organisation (CSIRO), DATA61, Brisbane, Australia
e-mail: Raja.Jurdak@data61.csiro.au

© Springer International Publishing AG 2018 137
C. Zachäus et al. (eds.), *Advanced Microsystems for Automotive Applications 2017*,
Lecture Notes in Mobility, DOI 10.1007/978-3-319-66972-4_12

1 Introduction

Future vehicles will utilize wireless communication networks to interact with other vehicles and road users in close proximity, roadside units like traffic lights and overhead displays at motorways, as well as the Internet. Thereby future connected vehicles will become part of the Internet of Things (IoT) and offer a plethora of beneficial services and applications to the users (e.g., the vehicle owner and driver) as well as the vehicle manufacturers (i.e., the OEM) and their suppliers. However, this high degree of connectivity also raises a wide range of new security threats as well as privacy concerns and will require a comprehensive security architecture. The importance of the latter was recently shown by different hackers attacking modern vehicles via their wireless interfaces (Valasek and Miller 2015; Foster et al. 2015).

Wireless over-the-air (OTA) software (SW) updates will be one of the key features of future connected vehicles and will allow adapting or upgrading the functionality of the vehicle or fixing bugs in the embedded SW installed on its electronic control units (ECU) remotely (Hossain and Mahmud 2007; Khurram et al. 2016). Such updates can be very beneficial for both the OEM (i.e., car manufacturer) as well as the end user (i.e., the vehicle owner) as they obviate the need for taking the vehicle to a service center to receive the latest SW version. However, OTA updates are very critical with repect to security as they require full access to the in-vehicle communication system to allow the installation of new SW images on all ECUs of a vehicle.

Because of their high potential and impact, automotive OTA updates have increasingly attracted the attention of the research community. Researchers have proposed various security architectures and concepts allowing trustworthy OTA updates for vehicles (Hossain and Mahmud 2007; Idrees et al. 2011; Nilsson and Larson 2008). In particular, existing work mainly focus on protecting a vehicle from unauthorized access and the injection of malicious SW. Other authors propose methods allowing secure and efficient SW updates performed locally in a service center or during vehicle assembly Steger et al. (2016), (2016). However, none of the aforementioned solutions address secure OTA distribution of SW from the OEM to all concerned vehicles while ensuring the privacy of the end user. Such a SW distribution process requires a highly scalable security architecture protecting the confidentiality as well as the integrity of the transferred data and furthermore retaining the privacy of the involved users.

In this paper, we propose an automotive security architecture utilizing Blockchain (BC) to tackle the implicated security and privacy challenges of future connected vehicles. Our BC-based security architecture can be utilized to perform OTA updates for smart vehicles remotely as well as to securely distribute the latest SW to service centers or vehicle assembly lines where the latest SW image is installed on the ECU of a vehicle locally. The proposed architecture ensures a secure as well as tamper-proof data exchange and protect the privacy of the end user. Thus, our architecture is not only applicable for protecting wireless automotive SW updates, but can also be utilized in a more general manner to secure a wide range of (future) automotive services.

The proposed architecture is evaluated using a proof-of-concept implementation of a wireless SW update system providing a secure as well as efficient communication between all involved parties: the SW provider (e.g., an automotive supplier) creating the latest SW, the OEM verifying, adapting (e.g., to fit specific vehicle variant) and finally distributing the SW, the cloud storage where the SW is stored, local SW update providers such as a service center, and the connected vehicle itself. We use this implementation to (i) show the applicability of our BC-based architecture for wireless automotive SW updates, (ii) analyze the packet overhead of the architecture due to the use of BC, (iii) highlight its advantages compared to centralized (e.g., certificate-based architecture), and (iv) evaluate the added latency compared to locally performed wireless SW updates.

2 Background

In this chapter, we present an overview of wireless SW updates and thereby describe the technical process as well as the scenarios where these updates can be most beneficial. Furthermore, we give some technical insights on BC, its initial usage as essential part of the crypto-currency Bitcoin, and explain required adaptions to use it in typical IoT as well as automotive applications.

2.1 Wireless Automotive Software Updates

A smart vehicle consists of dozens of ECUs performing different tasks such as controlling the window lifters, the engine, the windscreen wipers, etc. These ECUs are interconnected to each other via the in-vehicle communication system realized using different wired buses (e.g., CAN or LIN). A central vehicle gateway (CGW) is used to interconnect these bus systems as shown in Fig. 1. To wirelessly communicate with the vehicle and all its integrated ECUs, the vehicle has to be equipped with a Wireless Vehicle Interface (WVI) allowing full access to the in-vehicle communication system. Hence, a WVI is also required when a new SW version should be installed on one of the ECUs of a vehicle.

To perform a wireless SW update, the so-called Diagnostic Tester (DT), which possess all required keys to authorize the SW update and the new SW version, connects to the vehicle using its WVI. In the next step, the DT can use automotive diagnostic protocols such as Unified Diagnostic Services (UDS) to (i) initialize and authorize the SW update, (ii) transfer the binary to the ECU, and (iii) install it on the control unit. This procedure can be used locally in a service center, where the DT and the vehicle with a connected WVI are interconnected using a wireless local area

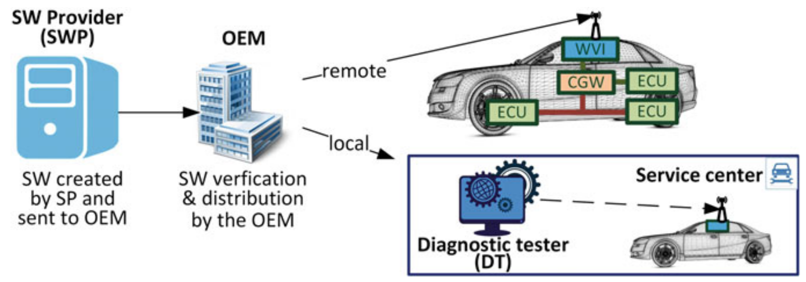


Fig. 1 A new SW version is created by the SW provider (SWP), verified and distributed by the OEM and finally installed on the concerned ECU of a vehicle

network such as Wi-Fi, but also remotely, where a diagnostic service of the OEM communicates with the WVI via the Internet. Both scenarios are sketched in Fig. 1. Local SW updates will mainly take place in service centers during vehicle maintenance, in the vehicle assembly line, and during vehicle development, where engineers will test and compare new SW versions.

A SW update for a vehicle already out in the field (i.e., already sold) is often required due to a (critical) bug in the automotive SW installed on one of its ECUs. The SW of an automotive ECU is often implemented by a supplier company and not by the OEM itself. Thus, in case of a bug fix, the latest (i.e., fixed) SW version is created by the supplier, hereinafter referred to as SW Provider (SWP), then sent to the OEM, and finally distributed to all concerned vehicles. The SW update procedure can either be directly handled between the OEM and the vehicle, or the latest SW is first sent to a service center, where the SW is then installed on the ECU locally. The described example is also sketched in Fig. 1.

2.2 *Blockchains*

BC technology was first introduced in 2008 as essential part of Bitcoin (Nakamoto 2008), the first cryptocurrency network. Since then, BC have been broadly used in nonmonetary applications (e.g., healthcare data exchange (Yue et al. 2016)) due to its security, privacy, and decentralization features. The secure nature of BC originates from the consensus algorithm employed for appending new blocks into the BC. The privacy of the involved users is ensured by utilizing changeable public keys (PK) representing the user. The BC is managed in a distributed fashion by all participating nodes which form an overlay network, thus not requiring any central management.

In a classical BC system the integration of a new block into the BC is done by the consensus algorithm employed to solve computational- and/or memory-expensive, cryptographic puzzles. Such classical systems, however, suffer from some significant limitations mainly caused by the extensive consensus algorithm, namely: (i) high resource consumption, and (ii) high latency.

These limitations are especially critical for a broad range of embedded applications, IoT services, and also resource-constraint ECUs. To tackle the aforementioned limitations, we developed LSB, a Lightweight Scalable BC (Dorri et al. 2017). In LSB, the resource-expensive consensus algorithm is replaced by a timing-based algorithm. Furthermore, LSB divides the network into clusters, which distributedly manage the public BC. Each cluster consists of numerous cluster members (CM) and is managed by one cluster head (CH). Each CH maintains a local copy of the BC and is interconnected to other CHs by the overlay network.

The BC consists of chained blocks containing different and application-specific transactions. To chain the blocks, the newest block contains the hash of the last chained block. A new block is created once the running pool, a local data structure of the CH containing new transactions (i.e., not included in a block yet), has reached a predefined size. In the next step, the new block is broadcasted to the other CHs using the overlay network and finally added to the BC of the other CHs. More detailed descriptions on this process can be found in Dorri et al. (2017).

3 Architecture Enabling Wireless Software Updates

Wireless automotive SW updates must be performed in a secure and dependable way as failed, malfunctioning, or malicious updates will significantly influence the operation of the vehicle and therefore the safety of the passengers. Besides the safety aspect with respect to the involved users (i.e., vehicle driver and owner), privacy considerations are also relevant in certain SW update scenarios (especially remote updates). A suitable automotive security architecture must ensure that a vehicle can receive the latest SW for its ECUs without exposing unrelated personal information about the vehicle and its users. Furthermore, such an architecture must protect the exchanged data at any time to (i) keep required (authorization) keys, mainly required to unlock the ECU, secret, (ii) maintain the confidentiality of the SW image, and (iii) ensure the integrity of the transferred data to avoid manipulation. These requirements are valid for the entire chain shown in Fig. 1: first the image is sent from the SWP to the OEM, second the SW image is forwarded to concerned vehicles and local SW update providers, and third the image is installed on the ECU (e.g., using a local wireless network in a service center).

While the local update scenario is already covered by security concepts such as Steger et al. (2016), the security of the SW distribution from a SWP to an OEM and then further to the vehicles is still an open issue. Some OEMs like Tesla (Gabe 2016) currently perform OTA updates using VPN tunnels between the OEM server infrastructure and the vehicle itself. Although this approach is suitable to protect the transferred data, it also requires a dedicated point-to-point link between the OEM and the vehicle which can potentially be critical with respect to the privacy of the end user. Other automotive security architectures use certificates to establish trust within the network (Woo et al. 2016; Aslam and Zou 2009) or (Mallissery et al. 2014). However, we believe that these centralized approaches are not suitable for highly distributed scenarios encompassing thousands of vehicles around the globe and therefore propose to use a BC-based automotive security architecture instead.

3.1 Blockchain-Based Architecture Securing Wireless Software Updates

Our BC-based architecture described in this section is able to fulfill the aforementioned security, privacy, and scalability requirements. In the following, we will focus on the secure SW image distribution from the SWP and the OEM to the target vehicles as shown in Fig. 1. Our architecture presented in Fig. 2, protects the transfer of SW images and also ensures the privacy of the involved users.

Our architecture is based on the design presented in Dorri et al. (2017). However, several adaptations were required to ensure that our architecture can meet the needs of the automotive domain. Additionally, different stakeholders and roles are involved in automotive scenarios compared to typical smart home applications.

As shown in Fig. 2, these roles are SW providers, OEMs, cloud storages (CS), and vehicular interfaces (VI) representing either smart connected vehicles or local SW update providers such as service centers or vehicle assembly lines.

The cloud storage is essential for our architecture as it serves as a repository for storing new SW images sent by the SWP or the OEM, and furthermore handles the data download to the VIs. In our architecture, the cloud storage provides a sophisticated authentication mechanism to ensure that only authorized entities can write, adapt, and download a specific SW image.

All aforementioned entities are interconnected using the overlay network. This clustered network allows unicast as well as broadcast data streams and is able to

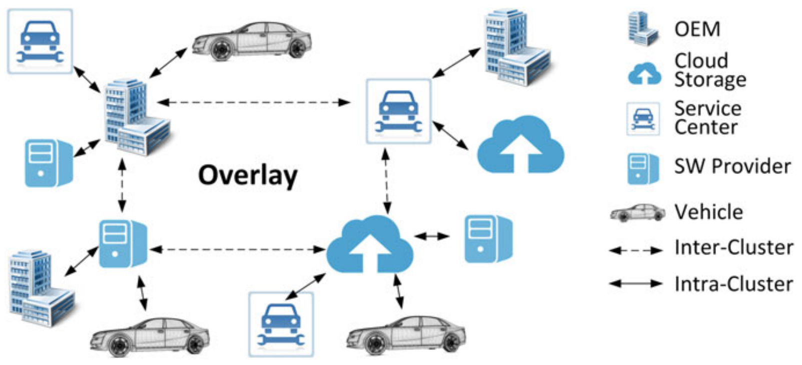


Fig. 2 The proposed BC-based architecture to securely interconnect all involved parties

provide suitable message flows for all different data exchanges involved in the SW distribution process. As described in Sect. 2.2, the overlay network basically interconnects different (e.g., geographical) clusters encompassing one CH and numerous CMs. Thereby, a CH acts as gateway for messages sent from a CM of a specific cluster (e.g., CM1 in cluster A) to CM or CH of another cluster (e.g., CM2 in cluster B). In our architecture, a CM and a CH can occupy specific roles such as acting as OEM, CS, or VI. However, as a CH besides its dedicated role also has to maintain its local BC, it is very unlikely that a vehicle would act as a CH due to its resource constraints and the fact that a vehicle is mobile and therefore not able to be connected to the overlay at all times. Our architecture uses two types of messages: (i) *transactions* and (ii) *blocks* for implementing OTA updates.

Transactions are used to initialize the BC system (*genesis transaction)* and to handle the SW distribution process (*update transaction*). The latter is required to inform the OEM (i.e., when sent by the SWP) as well as the concerned VIs (i.e., when sent by the OEM) about newly available SW. Update transactions, thus, play a vital role in our architecture as they contain required information for both the OEMs and the VIs: it contains the identities of the SWP as well as of the OEM (i.e., their PKs and signatures), the transaction ID (i.e., hash representation of the transaction), the ID of the previously created transaction (the genesis transaction for the very first update transaction), and the metadata field. The metadata field is used by the SWP and the OEM and includes information about the SW update itself as well as the location where the new image is stored.

A BC-block consists of several transactions as well as a link to the previous block that chain these blocks together. It is created every time the running pool of a CH reaches a predefined size. Each block has a unique ID and is signed with the private key of the creating CH. After creation, the block is broadcasted to all other CHs of the overlay for verification and then chained to the local BC.

3.2 Employing Our Architecture to Distribute New SW

The proposed automotive security architecture can be employed to securely distribute a new SW image to the target vehicles. We will now sketch the corresponding process, explain all involved steps and utilized message types, and show that our architecture is able to protect the entire process and all involved users.

In the vehicle assembly, the OEM will store its PK on each assembled vehicle and the vehicle will generate a secret key pair (e.g., an RSA key pair consisting of a private and a public key). Both the PK of the OEM and the key pair will be securely stored on the WVI in a tamper-proof storage. While a vehicle is assembled it can also create a genesis transaction, an initial transaction required to participate in the BC. This process can be highly OEM-dependent and will therefore not be described in more detail. However, we suggest that this transaction include information about the vehicle type (e.g., vehicle variant) and that the transaction is signed by the OEM. Or, as an alternative, a dedicated token including the aforementioned data and the signature of the OEM is created at this point in time. This transaction/token is required later to request new SW stored on a cloud storage.

A SW distribution process is triggered by a SWP when a new image is created potentially due to a necessary bug fix. Once the new SW is developed, the SWP will create a store request including the signature of the SWP and send it to the cloud storage. The latter will verify the request, locally initialize the process, and send a *store response* including the signature of the storage and a file descriptor required as reference for the data upload process back to the SWP. Please note that the SW can also be created by the OEM itself. In this case the OEM would upload the SW to the CS. The rest of the process is similar to that described above, except that the OEM will take over the tasks performed by the SWP. Once the data is stored on the cloud storage, the SWP creates an update transaction including information about the location of the image on the cloud storage, adds the PK of the concerned OEM, signs the transaction with its private key and finally broadcasts the transaction to the overlay network. Please note that the transaction is not valid yet as the second signature is missing and therefore it is not added in the running pool of the CHs.

In the next step the OEM receives the update transaction, verifies it, validates, and if required adapts the SW image stored on the cloud storage, and finally also signs the update transaction. The transaction which is now valid is again broadcasted to the overlay and locally stored by the CHs. The CHs will also send the transaction to all CMs in its cluster to inform vehicles about the new SW.

Finally, the valid transaction is received by the target VIs (i.e., vehicles or local SW update providers). After validating the transaction and parsing the metadata, the VIs will send a signed *download request* including the token signed by the OEM (e.g., stored on the WVI when a vehicle was assembled) to the cloud storage to receive the new SW version. The CS will validate the request, use the token to verify that the vehicle is applicable for the new SW, and is finally utilizing a unicast data stream to send the image to the VI, where the SW is installed.

4 Proof of Concept

In this section, we describe the implementation of the BC-based security architecture presented in the previous section. The implemented framework consists of two main building blocks (Fig. 3). First, the overlay network used by the SWP to send the latest SW to the OEM and employed by the OEM to distribute the verified SW version to the concerned vehicles as well as local distributors (e.g., service centers). Second, the local update process required to install the latest SW on the ECU. This step is executed by the WVI, which receives the latest SW either from the OEM via a remote connection or locally from a DT.

The implementation of the overlay encompasses the development of the CH and the CM as well as a local test suit to set up overlay test topologies (mainly on local host) consisting of several instances of the CHs and CMs. All the above is implemented in Java.

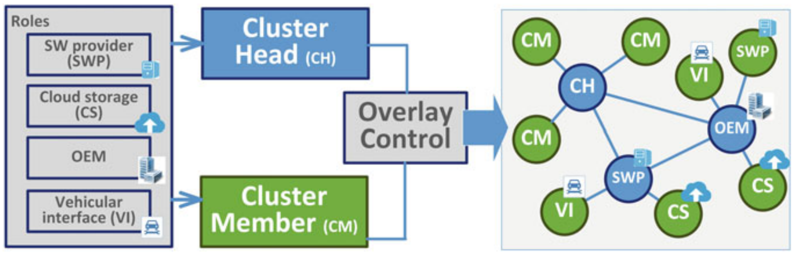


Fig. 3 Proof-of-concept implementation encompassing prototypes of the CHs and the CMs. The overlay control allows to design different test scenarios by utilizing serval CH and CM instances, assigning specific roles to these instances and defining the topology of the overlay

Our implementation allows us to evaluate different scenarios for different overlay topologies and numbers of VIs. We have also implemented a baseline system which is similar to the state of the art, wherein a dedicated Certificate Authority (CA) is employed to verify certificates used by the SWP as well as the OEM.

The local SW update building block is based on the wireless automotive SW update framework we presented in Steger et al. (2016). This framework allows to perform local SW updates in a secure and dependable way by employing IEEE 802.11 s to interconnect the DT and the vehicle with an integrated or connected WVI (via OBD).

Our WVI prototype, as shown in Fig. 4a, consists of a BeagleBone Black board (BBB), an additional communication cape (i.e., a printed circuit board allowing the BBB to connect to a vehicle via CAN/OBD), and the corresponding SW implementation (Java and C). The DT prototype is also realized in Java and can therefore be used on a normal PC but also on a BBB. Both prototypes provide different SW update mechanisms and allow different diagnostic functions.

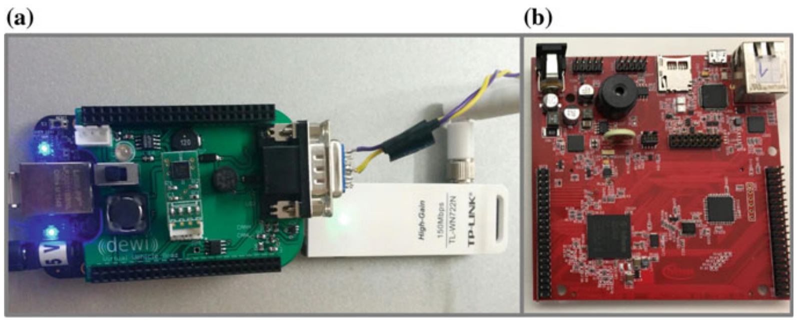


Fig. 4 **a** The WVI prototype based on a BeagleBone Black and our developed communication cape; **b** target ECU: Infineon AURIX ECU in the AURIX application kit TC277 TFT

The vehicles' WVI and the DT are interconnected using an IEEE 802.11 s mesh network. We chose this protocol as the mesh characteristics of an IEEE 802.11 s network increases the flexibility as well as the reliability of the network due to its multi-hop capability and the resulting redundancy.

As target ECU for the SW update we use an Infineon AURIX ECU, an automotive multi-core ECU, assembled in the AURIX application kit TC277 (Fig. 4b).

5 Evaluation

We used our proof-of-concept implementation to show the applicability of our architecture to fulfill the needs of an automotive OTA SW update system. Thereby, we evaluated the (packet) overhead when using BC and compared the duration of the BC-based SW distribution with the time required to install a SW update locally on an ECU. Furthermore, we compared our architecture with the baseline certificate-based system outlined in Sect. 5. We used this evaluation to compare the total number of packets exchanged as well as the latency incurred in the SW distribution process.

5.1 *Overhead Due to the Use of Blockchains*

In the first evaluation step we analyzed the overhead added by the BC. Therefore, we collected the number of exchanged packets and grouped them into data-related, BC-related, and packets required to initialize the system. The overhead is affected by the number of VIs, the size of the binary, and the number of performed updates. Our evaluation for a 32 KB binary and 100 SW updates per VI reveals an added overhead of 3.4% for 20 VIs connected to the overlay and up to 7.3% if only 1 VI is updated. Neglecting the initialization packets, the overhead is only 3.3% for twenty VIs.

5.2 *Latency Comparison: Local SW Update Versus SW Distribution Using BC*

In this experiment, we compared the latency added by the BC-based SW distribution and the latency of the SW update process itself. For this, we measured the latency of a local wireless SW update using the framework presented in Steger et al. (2016) as well as the time required for the last step of a remote SW update, where a new SW update is installed on an ECU by the WVI using the vehicle bus system.

The results are presented in Table 1 and show that the installation of a new image on the ECU using the wired in-vehicle bus system takes more than five times

Table 1 Comparison of the latency of the SW distribution, a local wireless SW update, and the installation of a new SW image on an ECU performed by the WVI

SW distribution	Wireless local update	WVI installation
2682.3 ± 8.3 ms	16271.0 ± 323.4 ms	13831.7 ± 228.3 ms

longer than the SW distribution from an emulated SWP to the VI using our BC-based system and that the SW distribution process using our experimental setup is about six times faster than the local SW update process. Note that the SW distribution was performed using a LAN network architecture. Therefore, the SW distribution latency does not include any additional latency caused by typical Internet links. The latency of local SW update and the SW installation can also vary depending on the used ECU, the employed security mechanisms, etc.

5.3 Comparison of BC- and Certificate-Based Approaches

We evaluated the number of exchanged packets as well as the latency of our BC-based system compared to a certificate-based approach. Therefore, we used an overlay network consisting of up to ten HW nodes (several BBBs, Raspberry Pi3's, and a Laptop) interconnected by the overlay network and performed measurements using different network topologies and different numbers of VIs.

The evaluation results presented in Figs. 5 and 6 show that both approaches have quite similar properties with respect to the added latency as well as the total number of exchanged packets and that our BC-based approach is slightly better than certificated-based approach in both aspects.

The performed experiments and measurements showed that (i) BC approximately add 3% packet and 14% latency overhead compared to a pure OTA SW update, (ii) BC has lower latency and uses fewer packets than a certificate-based system, and (iii) show the applicability of our architecture for automotive applications.

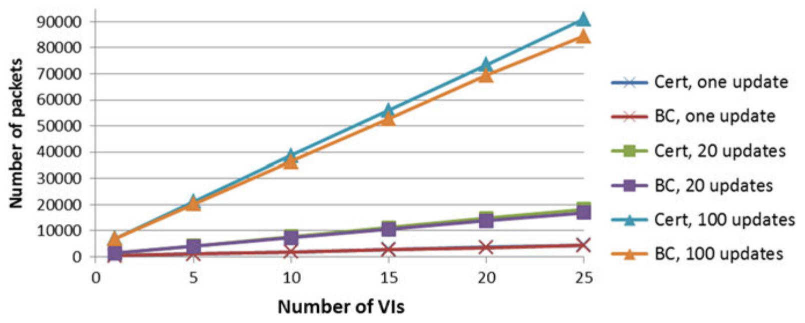


Fig. 5 Exchanged packet count comparison with respect to the number of VIs and number of updates

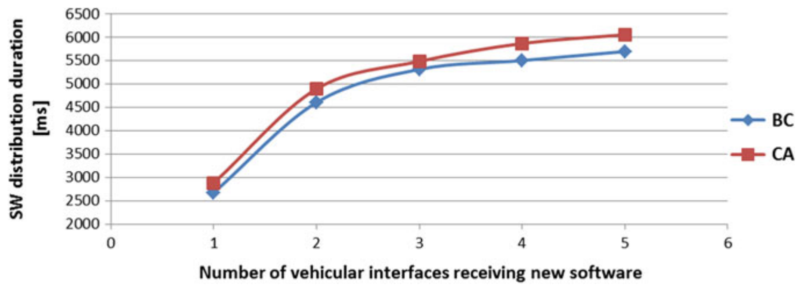


Fig. 6 Comparison of the SW distribution duration for different numbers of involved VIs

6 Conclusion

In this paper, we proposed a security architecture based on BC for smart-connected vehicles able to support a broad range of (future) automotive applications and services. Our architecture provides a secure and trustworthy interconnection between all involved parties while ensuring the privacy of the involved users. We evaluated our architecture using a proof-of-concept implementation of a wireless SW update system and use the latter to show the applicability of our architecture as well as its benefits compared to a certificate-based system. We plan to further refine our architecture, to improve our implementation, and to perform more detailed evaluations employing more nodes in the next months.

References

Aslam B, Zou C (2009) Distributed certificate and application architecture for VANETs. In: IEEE military communications conference, pp 1–7

Dorri A, Kanhere S, Jurdak R (2017) Towards an optimized blockchain for IoT. In: Proceedings of the second international conference on internet-of-things design and implementation (IoTDI '17). ACM, pp 173–178

Foster D, Prudhomme A et al (2015) Fast and vulnerable: a story of telematic failures. In: USENIX workshop on offensive technologies

Gabe N (2016) Over-the-air updates on varied paths, automotive news

Hossain I, Mahmud S (2007) Analysis of a secure software upload technique in advanced vehicles using wireless links. In: Intelligent Transportation Systems Conference, pp 1010–1015

Idrees M, Schweppe H et al (2011) Secure automotive on-board protocols: a case of over-the-air firmware updates. Lecture Notes in Computer Science. LNCS, vol 6596, pp 224–238

Khurram M, Kumar H et al (2016) Enhancing connected car adoption: security and over the air update framework. In: IEEE world forum on internet of things (WF-IoT), vol 3, pp 194–198

Mallissery S, Pai M et al (2014) Improving the PKI to build trust architecture for VANET by using short-time certificate mngt. and Merkle Signature Scheme. In: Asia-Pacific conference on computer aided system engineering, pp 146–151

Nakamoto S (2008) Bitcoin: a peer-to-peer electronic cash system. http://www.bitcoin.org/bitcoin.pdf

Nilsson D, Larson U (2008) Secure firmware updates over the air in intelligent vehicles. In: IEEE conference on communications, pp 380–384

Steger M, Karner M et al (2016) Generic framework enabling secure and efficient automotive wireless SW updates. In: IEEE international conference on emerging technologies and factory automation (ETFA), vol 21, pp 1–8

Steger M, Karner M et al (2016) SecUp: secure and efficient wireless software updates for vehicles. In: IEEE conference on digital system design (DSD), pp 628–636

Valasek C, Miller C (2015) Remote exploitation of an unaltered passenger vehicle, White Paper, p 93

Woo S, Jo H et al (2016) A practical security architecture for in-vehicle CAN-FD. IEEE Trans Intell Transp Syst 17:2248–2261

Yue X, Wang H et al (2016) Healthcare data gateways: found healthcare intelligence on blockchain with novel privacy risk control. J Med Syst 40:1–8

Paper F

M. Steger, C.A. Boano, K. Römer, M. Karner, J. Hillebrand, and W. Rom. **CESAR: a Testbed Infrastructure to Evaluate the Efficiency of Wireless Automotive Software Updates.** *In Proceedings of the 20th International Conference on Modeling, Analysis and Simulation of Wireless and Mobile Systems (MSWiM).*, pages 311–315. Miami, USA. November 2017.

Summary. This paper proposes CESAR, a configurable testbed infrastructure that allows to evaluate the effectiveness and efficiency of wireless automotive software updates in an automated and repeatable way. CESAR allows to investigate the software update procedure in its entirety, to emulate different software update scenarios, and to evaluate the impact of different network as well as security configurations on the updates efficiency. The proposed testbed infrastructure can be further used to analyze different ECU types, software update techniques (e.g., the parallel or partial transfer of a firmware), and wireless communication standards (e.g., the use of single-hop or multi-hop networks). A series of case studies presented in this paper are illustrating how CESAR can be used to evaluate the impact of different security configurations, update techniques, and network protocols on the efficiency of an automotive software update process.

My contributions. I am the main author of this paper and developed the entire testbed infrastructure as well as carried out the presented case studies using CESAR. I wrote the vast majority of this paper in collaboration and discussion with the co-authors, who provided detailed feedback regarding the design of the testbed infrastructure as well as the corresponding descriptions. Carlo Boano was significantly supporting this paper by improving the structure of the paper, by helping me defining several case studies, and by (re-)writing and improving certain sections of the paper (i.e., related work and introduction).

©2017 ACM.
ISBN: 978-1-4503-5162-1.
DOI: 10.1145/3127540.3127580.

Link: https://dl.acm.org/citation.cfm?id=3127580.

- On the author’s own home page;
- On the author’s institutional repository;
- In any repository legally mandated by the agency funding the research on which the work is based.

CESAR: A Testbed Infrastructure to Evaluate the Efficiency of Wireless Automotive Software Updates

Marco Steger[‡], Carlo A. Boano[†], Kay Römer[†], Michael Karner[‡], Joachim Hillebrand[‡] and Werner Rom[‡]
[‡]Virtual Vehicle Research Center, Graz, Austria
[†]Institute for Technical Informatics, Graz University of Technology, Austria
{marco.steger, michael.karner, joachim.hillebrand, werner.rom}@v2c2.at – {cboano, roemer}@tugraz.at

ABSTRACT

Connected vehicles allow to update the software (SW) running on their integrated electronic control units (ECUs) over-the-air. Such updates are complex procedures that involve several steps, such as the authentication with a remote device, the secure and reliable wireless transfer of the new binary, as well as its installation and verification on the target ECU. Each of these aspects affects the efficiency of the *entire* SW update process, and it is important to evaluate the impact of different solutions on the functionality of a vehicle and to compare their performance on real hardware. In this paper we present CESAR, a configurable testbed infrastructure that allows to evaluate the efficiency of an automotive SW update system in a highly automated way. CESAR allows to specify different update mechanisms, security configurations, wireless protocols used for the data transfer, and to carefully define the scenario of interest (i.e., pin down the number of wireless vehicle interfaces, the network topology, and the target ECU). Furthermore, CESAR can be used to measure the efficiency of a SW update on real hardware, and to derive insights about the weaknesses of a system under test or about the interaction of a specific SW with a given ECU.

KEYWORDS

Automotive Software, IEEE 802.11s, OTA Updates, Testbeds.

1 INTRODUCTION

The ability to wirelessly connect a vehicle to the Internet, to the road infrastructure, or to other vehicles, allows vehicle manufacturers (OEMs) to provide a plethora of new safety functions, comfort features, and services. Among others, automotive OEMs have the possibility to remotely diagnose a vehicle, as well as to install new SW on the ECUs over-the-air, which allows to reduce warranty costs [12]. Besides enabling performance improvements and bug fixes without the need of expensive vehicle recalls, wireless SW updates allow OEMs to upgrade or enable new features remotely.

The use of over-the-air (OTA) SW updates is not only limited to the remote download of up-to-date SW directly by the car owners (e.g., Tesla OTA updates [4]), but can also be exploited in several other stages of a vehicle's lifetime: from the vehicle development and the manufacturing stage on the assembly line, to the maintenance in a service center [13]. In all these scenarios, the vehicle uses its wireless vehicle interface (WVI) to connect to a diagnostic tester (DT) device holding the new SW binary, authorization keys, as well as other information that is required to perform the update. The update procedure itself can be conducted using *automotive diagnostic protocols* such as Unified Diagnostic Services (UDS) [1].

Due to their potential impact, OTA updates have increasingly attracted the attention of several researchers, who started analyzing the vulnerabilities of automotive eco-systems [6], and providing solutions to orchestrate secure SW updates [14]. Among others, the research community has proposed architectures to protect a vehicle from the injection of malicious SW [9, 10], and techniques to ensure reliable (wireless) inter/intra-vehicle communication [15, 17]. Most of the existing works, however, focus only on *single aspects* of an automotive SW update and not on the *entire* update process.

Need to evaluate SW updates in their entirety. The update procedure involves multiple steps ranging from the authentication with the DT and the wireless data transfer, to the installation and verification of the new binary on the target ECU. All of these aspects are interconnected and affect the overall *efficacy* and *efficiency* of a SW update, which should be always studied in its entirety. The latter requires a deep investigation of the main aspects affecting the efficiency of a SW update, such as: i) the *wireless network topology* and the number of involved nodes, ii) the applied *security configuration*, iii) the employed *SW update mechanism*, and iv) the *target ECU* and the properties of the connection to the WVI.

Need for suitable automotive testbeds. All these aspects must be evaluated in a systematic and repeatable way on real hardware (HW) to study their inter-dependency and to show the applicability of the tested SW update system. Towards this goal, it is necessary that the testbed supports not only a number of WVIs, but also their connection to one or more ECUs using automotive standard HW and SW interfaces, as well as means to install and verify the SW running on the ECU by means of diagnostic standards.

Our contributions. In this paper we present CESAR, a Configurable testbed infrastructure that allows to evaluate the effectiveness and Efficiency of wireless automotive Software updates in an Automated and Repeatable way. CESAR allows to investigate the SW update procedure in its entirety, to emulate different SW update scenarios (e.g., SW updates in a service center or in the assembly line), and to evaluate the impact of different network as well as security configurations on the update's efficiency. The proposed testbed infrastructure can be further used to analyze different ECU types, SW update techniques (e.g., the partial transfer of firmware), and wireless communication standards (e.g., multi-hop networks).

MSWiM '17, November 21–25, 2017, Miami, FL, USA
© 2017 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN 978-1-4503-5162-1/17/11…$15.00
https://doi.org/10.1145/3127540.3127580

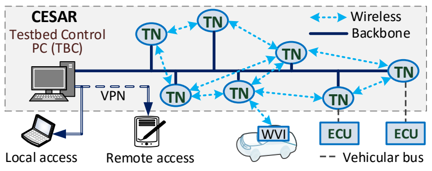


Figure 1: CESAR architecture: TNs interconnected via a backbone network allowing to connect ECUs or vehicles.

CESAR provides configuration profiles containing different node configurations (emulating different real-world scenarios), sets of parameters (e.g., key length, vehicle bus bit-rate), as well as SW update techniques. After describing its design and implementation in the next section, we show in Sect. 3 a series of case studies illustrating how CESAR can be used to evaluate the impact of different security configurations, update techniques, and network protocols on the efficiency of an automotive SW update process.

2 CESAR: DESIGN AND IMPLEMENTATION

We describe next the design and implementation of CESAR, starting from the general requirements of such a testbed infrastructure.

2.1 Testbed Requirements

A proper testbed infrastructure should support the evaluation of the *entire* automotive SW update process and allow to study the impact of different aspects on its efficiency while reducing manual intervention and allowing remote access. The employed testbed nodes must be able to support different roles (i.e., act as WVI, as DT, or as rogue node) and should be connectable to one or more ECUs from different vendors (which requires automotive HW/SW interfaces, as well as diagnostic protocols on top). Furthermore, the testbed should be able to scale up to 100 nodes while providing multiple *configuration profiles* that allow the user to choose between different network topologies and wireless communication stacks (e.g., IEEE 802.11n or 11s). These configuration profiles should include different security configurations and allow the user to choose between different security parameters, such as the authentication scheme or the key length. Ideally, also the installation effort is kept to a minimum by reusing existing network infrastructures.

2.2 Testbed Architecture

The architecture of CESAR is shown in Fig. 1: at the heart of CESAR are several testbed nodes (TNs) connected to each other wirelessly and to a testbed control PC (TBC) through a wired back-channel.

Testbed nodes. Each TN is configurable and can hence assume different roles within the testbed: it can act as a DT, WVI, relay node, or even as rogue node – a node that is compromised by an attacker. Depending on the assigned role, a TN runs a dedicated SW implementation on top of a given HW platform. The latter allows to connect a TN to an ECU using automotive bus systems (e.g., CAN or FlexRay) and easily install a new software.

ECU connection. By using automotive standards, CESAR allows to connect ECUs of different manufacturers and types, hence giving a user the ability to install SW on an ECU and to verify the success of an update procedure. In the simplest case, where different ECU SW versions periodically send CAN frames with different IDs, the verification is done on the TN acting as WVI by monitoring the CAN bus. This simple but efficient mechanism can be used for all target ECUs, even if there is no way to adapt the bootloader or flashing mechanism of the ECU. For more detailed tests, advanced features like computing the hash of the entire memory on the ECU after a SW update can be very beneficial. Therefore, we provide such features on our main target ECU, which allows CESAR to monitor the state of the ECU (via CAN) while a SW update is performed.

Configurability. CESAR provides configuration profiles allowing a simple configuration of the testbed and all its nodes. A configuration profile is a set of configuration files that can contain i) specific security and/or network configurations, ii) a certain node setting allowing to emulate specific real-world SW update scenarios, iii) a set of system parameters such as the vehicle bus bit rate or the employed authentication mechanism, and iv) specific SW update mechanisms. By utilizing the TBC, a developer can easily switch between different experimental settings and redo an experiment later by selecting this configuration profile again.

Remote monitoring. The TBC also allows other devices to access the testbed remotely. We developed a GUI that allows to monitor the state of the TNs, to set basic parameters (e.g., the wireless channel), to individually control TNs (e.g., reset TNs), and to select specific configuration profiles for planned experiments.

Prototype testing. CESAR allows to analyze the performance of different SW versions in a highly automated manner by storing and administrating all developed SW prototypes in a centralized repository. A developer can choose a specific SW version by using a specific configuration profile: prototypes are then automatically distributed to the TNs and locally configured.

SW architecture. The SW architecture of CESAR is shown in Fig. 2, and encompasses the implementations of different testbed features, the SW update system under test, and the interfaces used to interconnect the devices and the implemented prototypes. The testbed-specific SW blocks on the TBC are needed to i) (remotely) control the testbed and the running experiments, ii) retrieve specific versions of the developed DT/WVI prototypes from a GIT repository, and iii) automatically collect, pre-process and store the results of the experiments. On the TNs, testbed-specific SW blocks are required to i) assign the role of the TN, ii) locally configure the TN (e.g., vehicle bus bit-rate when connecting to an ECU), iii) enable/configure local parts of the testbed (e.g., monitoring the vehicle bus or storing debug information w.r.t. the wireless network), and iv) collect the results of an experiment and send it to the TBC.

2.3 Implementation

CESAR currently makes use of twelve TNs deployed on the ceiling of our office building, covering an area of approximately 350 m^2.

Testbed nodes. Each TN consists of a *BeagleBone Black* (BBB) board running Debian Linux and a *TL-WN722 Wi-Fi stick*, connected via USB and enabling wireless connectivity between the TNs (either IEEE 802.11n or 11s). We connect each BBB to a *custom-made PCB board* allowing the TNs to support up to two CAN connections at the same time [14]. UDS is used to perform the actual SW update procedure. Given the popularity of CAN and UDS, CESAR allows to connect a TN to almost any ECU on the market.

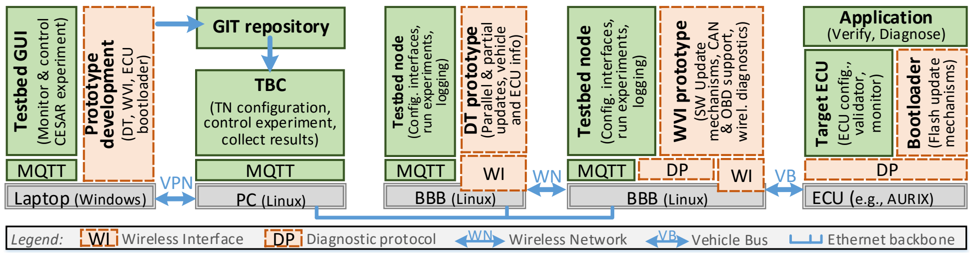

Testbed GUI (Monitor & control CESAR experiment)
Prototype development (DT, WVI, ECU bootloader)
GIT repository
TBC (TN configuration, control experiment, collect results)
Testbed node (Config. interfaces, run experiments, logging)
DT prototype (Parallel & partial updates, vehicle and ECU info)
Testbed node (Config. interfaces, run experiments, logging)
WVI prototype (SW Update mechanisms, CAN & OBD support, wirel. diagnostics)
Application (Verify, Diagnose)
Target ECU (ECU config., validator, monitor)
Bootloader (Flash update mechanisms)
MQTT
MQTT
MQTT
MQTT
WI
DP
WI
DP
VPN
WN
VB
Laptop (Windows)
PC (Linux)
BBB (Linux)
BBB (Linux)
ECU (e.g., AURIX)
Legend: WI Wireless Interface DP Diagnostic protocol WN Wireless Network VB Vehicle Bus Ethernet backbone

Figure 2: Software architecture. Main blocks of the testbed (green blocks, single solid line) and of the update system under test (orange blocks, dashed line). Interface are shown in blue and the employed devices using a gray block and double solid line.

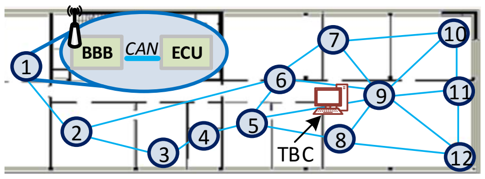


Figure 3: Position of the twelve nodes used in our testbed.

We connected the testbed nodes to two different types of ECU: the *Volvo FlexECU*, a prototype ECU used by Volvo Trucks and other automotive OEMs to test new vehicular features, and the *Infineon AURIX ECU*, a multi-core ECU used in various research and industry projects. In contrast to the Volvo FlexECU, which we had to use as black-box device without the possibility to develop our own ECU application SW, the AURIX comes with a free development tool-chain and can be powered via the I/O pins of the BBB board. This allowed us to easily connect the AURIX to the TN and to have full control on both the ECU application SW and the bootloader.

Backbone network and TBC. Each BBB board is connected to the backbone network using its Ethernet interface. To minimize the cabling effort, we exploit the existing 100 Mb/s LAN infrastructure of our office building. The TNs are decoupled from the rest of the company network infrastructure by using a dedicated subnet supporting up to 250 static IPv4 addresses. Each BBB board uses its Ethernet interface as a back-channel to communicate with the TBC. The latter is a desktop computer running Debian Linux equipped with a dual Ethernet card to connect to both the TNs and the office network infrastructure. This allows the TBC to access the Internet and other company services, such as a GIT server for source code management. To interact with the TNs, the TBC runs a Message Queue Telemetry Transport (MQTT) server. The MQTT publish-subscribe protocol allows to address all TNs at once (e.g., to start a measurement) and to configure a TN individually (e.g., assigning a specific role). Our implementation also allows to access the TBC remotely by using a VPN connection to the company network.

3 CASE STUDIES

We illustrate next a few use cases showing how CESAR can be used to study the efficiency of automotive wireless SW updates and to analyze the impact of different system configurations.

3.1 Impact of different Security Mechanisms

We first use CESAR to analyze the impact of different security mechanisms and key lengths on the duration of a SW update.

Impact of network and application layer security. We select different security configuration profiles at both application and network layer, and let CESAR automatically configure the testbed nodes with the specified security settings, while measuring the update duration and the detailed, per-step latency, using the logging ability of the DTs and the WVIs. We employ security mechanisms on the application layer implemented in SW utilizing the Java Bouncy Castle, whereas we use simultaneous authentication of equals (SAE) to protect the network layer[1]. We use the testbed deployment shown in Fig. 3 and configure node 9 to work as a DT and node 8 to act as WVI with a Volvo FlexECU connected via CAN. We choose this configuration, as it ensures a direct stable link between the DT and the WVI[2]. We perform a wireless SW update of a binary of 445 kB and measure the duration of the following steps ten times: i) Init: including WVI discovery, connection and authentication process between WVI and DT, and SW update initialization and authorization step on the ECU; ii) Upload: wireless data transfer from DT to WVI; iii) Download: data download via CAN and validation of the installed SW on the ECU.

Table 1 shows the measured overall duration and the per-step-latency w.r.t. the used security mechanisms and reveals that the data transfer from the WVI to the ECU via CAN takes the largest portion: about 75% of the overall duration with all security features enabled, and up to 87% if these mechanisms are disabled. The results also expose that the security functions have a significant impact on the update duration: plus 18.5% when all mechanisms are enabled.

Impact of the key length on the update duration. We use the aforementioned experimental setup and configure CESAR to disable the security mechanisms on the network layer. Different configuration profiles are then used to evaluate the impact of the key length on the update duration. Specifically, we use different key lengths for both the RSA-based authentication and AES-based data encryption, and perform 10 sequential wireless SW updates using the AURIX ECU for each configuration. Table 2 shows the duration of a SW update depending on the employed key length: varying the key length of the RSA-based authentication has a stronger impact on the update duration than different AES encryption key lengths.

[1]For more details on the utilized security mechanisms, we refer the reader to [13].
[2]CESAR would also allow a more difficult setup with links of intermediate quality.

Table 1: Duration in ms of the different wireless SW update steps depending on the employed security mechanism.

Security	Total	Init&Auth.	Upload	Download
Appl. + Net.	49195.4	4782.3	6585.4	37817.7
Application	47167.1	4859.7	5414.2	36885.3
Network	43745.6	2277.7	3756.0	37703.3
None	41528.6	2277.9	2445.0	36797.5

Table 2: Impact of the key length on the update duration

RSA	AES	Duration	Delta
1024	128	16271.0 ± 323.4 ms	-
1024	256	16375.1 ± 222.3 ms	104.1 ms (+0,6%)
2048	128	18342.7 ± 357.4 ms	2071.7 ms (+12.7%)
2048	256	18359.1 ± 292.3 ms	2088.1 ms (+12.8%)

Table 3: Update duration w.r.t different update mechanisms.

Traditional	Parallel	Partial
20768.8 ± 882.4 ms	25881.8 ± 324.5 ms	3570.7 ± 1189.3 ms

3.2 Impact of different ECU Hardware

We illustrate CESAR's ability to support ECUs of different vendors by connecting a WVI device to both a Volvo FlexECU and an AURIX ECU via CAN (at 500 kb/s). We employ node 9 as DT and node 8 as WVI according to the testbed deployment shown in Fig. 3. As the Volvo FlexECU can only be used as a black-box, we create a dummy application for the AURIX that has the same binary size as the one available for the Volvo FlexECU (i.e., a size of 445 kB). We then run twenty wireless SW updates on each of the two ECUs, and let CESAR measure their average duration a discussed earlier.

The gathered results show that the AURIX ECU can be updated about three times faster than the Volvo FlexECU: the SW update takes indeed 20.77±0.88 and 48.68±0.81 seconds on the AURIX and the FlexECU, respectively. This is due to (i) the higher CPU power and faster storage modules of the AURIX ECU, and (ii) the fact that the FlexECU uses a two-stage update procedure encompassing a secondary bootloader and the application SW itself.

3.3 Impact of different Update Techniques

CESAR can also be used to evaluate the efficiency of different SW update mechanisms: traditional, parallel, and partial updates.

Parallel updates. We first compare the duration of the update process when using traditional SW updates sequentially with the duration of a *parallel* SW update. A parallel SW update is performed on two or more ECUs integrated in two or more vehicles at the same time. For an experiment within CESAR this means that new SW is installed on several (in this particular experiment two) ECUs at the same time by one DT (node 9). Therefore, each ECU is connected to a testbed node acting as WVI (nodes 8 and 11) and the SW update is first done sequentially (first node 8 is updated and then node 11) and then in parallel, meaning on both ECUs at the same time.

Table 3 shows that the overall duration of one parallel SW update (for two ECUs) is increased by about 25% compared to a traditional wireless SW update (for one of the ECUs). This overhead is due to the fact that not all steps of a SW update can be done in parallel. However, carrying out parallel SW updates is significantly faster (approximately 75% quicker) compared to a sequential updates.

Table 4: Update duration w.r.t. different network protocols.

802.11s duration	802.11n duration	TN	RSSI
16590.4 ± 286.1 ms	16545.0 ± 360.8 ms	6	-83 dBm
16943.0 ± 453.1 ms	16504.7 ± 238.4 ms	4	-87 dBm
16918.2 ± 518.4 ms	21274.8 ± 4632.9 ms	2	-90 dBm
17620.1 ± 1696.6 ms	Unreachable	1	>90

Partial updates. A SW update is often only changing specific parameters of an ECU, leaving most of the remaining SW untouched. For this reason, it may be advisable to only update the changed SW portion, instead of the entire binary. We compare next the duration of a traditional SW update with a custom implementation of a *partial* update in which only the portion of code that has changed is installed. We prepare two SW applications with a parameter field stored in a dedicated memory section of the AURIX ECU of size 1 kB: this parameter field is the only difference between the binaries. When utilizing a partial SW update, only this section is transferred to the ECU. Instead, when using a traditional SW update, the entire binary (of size 445 kB) needs to be transferred. The case study was performed using node 9 as DT and node 8 as WVI connected to an AURIX ECU. We performed twenty SW updates using both the traditional and the partial SW update mode.

Table 3 highlights that, as expected, the SW update duration is significantly reduced: the partial update is about six times faster. This decrease in duration of 83% is especially due to the lower amount of data transferred from the WVI to the ECU via CAN.

3.4 Impact of different Network Protocols

In this case study, we evaluate the efficiency of SW updates of two different wireless protocols: IEEE 802.11n and IEEE 802.11s. The key difference between these two protocols is the ability to build mesh networks: IEEE 802.11n is the traditional, access point based Wi-Fi protocol. Instead, IEEE 802.11s allows to form multi-hop networks increasing the reliability and availability of the entire network.

In our experimental setup illustrated in Fig. 3, we select node 11 as DT and different TNs (nodes 1, 2, 4, and 6) connected to an AURIX ECU. For the IEEE 802.11n evaluation, the DT node is also acting as wireless access point. It is important to highlight that, in this configuration, the links between 11 and the other TNs are of different quality. Furthermore, node 1 is the furthest away, and is not in the communication range of node 11. As a result, when employing IEEE 802.11n, no communication can be established between the two nodes. In contrast, when using IEEE 802.11s, the nodes can decide to hop through additional nodes to use only very good links and maximize the reliability of communications.

We then perform ten SW updates for each configuration and let CESAR measure the SW update duration (Table 4). For good links (node 4 and 6), both protocols nearly exhibit the same performance. For node 4, IEEE 802.11n, on average, is about 400 ms faster than IEEE 802.11s. The experimental results show an average path length of 1.06 hops when using IEEE 802.11s. This means that IEEE 802.11s has employed some multi-hop paths during the SW update process (due to lost packets) leading to a slightly increased packet latency. In case of intermediate links (node 2), IEEE 802.11s outperforms IEEE 802.11n by a factor of 25%.

3.5 Connectivity issues of IEEE 802.11s

CESAR can also be used to investigate the connectivity of wireless nodes and thereby, as presented in this case study, to reveal scalability issues. When configuring CESAR to use IEEE 802.11s, we observed that some TNs were not reachable by other nodes, despite being in close proximity. The observed issue is critical as typical SW update scenarios can encompass several vehicles in a dynamic environment frequently joining and leaving the network. To analyze the problem, snapshots of the connectivity in the network (i.e., link and path information) were captured using CESAR. These tests reveal two major problems of the default open11s implementation caused by its limited neighbor table size:

Inefficient network structure. In IEEE 802.11s, peer links are established between two nodes if i) the nodes can *hear* each other and ii) the nodes have a free spot in their neighbor table. Hence, the network topology of 11s is mainly influenced by the sequence of nodes joining the network (and not if a node is close or far away) and thus leads to inefficient links affecting the network performance.

Isolated node. In the worst case the limited neighbor table size (e.g., size=3) can even lead to isolated nodes: a node (e.g., node 5) willing to join an already established network, will fail to connect as the other nodes (e.g., nodes 1 to 4) already have three neighbors stored in their neighbor table. The nodes will decline the join requests and node 5 will be isolated from the network.

We solved this issue by adapting the latest open11s implementation: i) adding new messages to inform the network about isolated nodes, and ii) implementing algorithms to solve the *isolated node problem*. The adapted open11s version can be chosen and configured by CESAR besides the default open11s version.

4 RELATED WORK

In this section, we summarize the body of existing works focusing on wireless automotive SW updates and compare the functionality of existing automotive testbeds to the ones offered by CESAR.

Wireless automotive SW updates. Most of the work on automotive OTA updates has focused on security aspects and proposed security architectures protecting a vehicle from malicious updates [9, 11] or investigated remote SW updates [7, 10]. These works, however, do not consider other scenarios where updates are performed locally (such as within a service center), nor allow testing of advanced update mechanisms such as the parallel transfer of a binary. Solutions are also often evaluated only through simulation [5, 11] or formal methods [9, 10, 13], and very few systems are evaluated on real HW. In this and our previous works [8, 14], automotive ECUs are used to evaluate a SW update framework, verify the update process, and compare different update mechanisms.

Automotive testbeds. Several infrastructures have been proposed to test automotive SW updates. Drolia et al. [3] have designed a testbed consisting of several automotive ECUs interconnected by CAN. Although the testbed provides a basic SW update function, it is not capable of evaluating the entire wireless SW update process. In [16] and [2], authors have proposed Vehicle-to-Vehicle testbeds (either indoor [16] or outdoor [2]) to simulate different V2V scenarios. However, these testbed do not support automotive ECUs and cannot be used to evaluate any aspect w.r.t wireless SW updates.

5 CONCLUSIONS

In this paper we propose CESAR, a testbed infrastructure that allows to investigate the efficiency and dependability of an entire wireless automotive SW update process. After describing the testbed architecture and design, we show through a series of case studies that CESAR allows to derive insights about the impact of different security configurations, update techniques, and network protocols on the efficiency of an automotive SW update process. In the future, we plan to use CESAR to evaluate the reliability of IEEE 802.11s and to run different attacks on the SW update framework presented in [13], in order to evaluate its robustness and expose its weaknesses.

Acknowledgments. This work was partially funded by the SCOTT project. SCOTT (http://www.scott-project.eu) has received funding from the Electronic Component Systems for European Leadership Joint Undertaking under grant agreement No 737422. This joint undertaking receives support from the European Unions Horizon 2020 research and innovation programme and Austria, Spain, Finland, Ireland, Sweden, Germany, Poland, Portugal, Netherlands, Belgium, Norway. SCOTT is also funded by the Austrian Federal Ministry of Transport, Innovation and Technology (BMVIT) under the program "ICT of the Future" between May 2017 and April 2020. More information at https://iktderzukunft.at/en/. The authors also acknowledge the financial support of the COMET K2 Program of the Austrian Federal Ministries BMVIT and BMWFW, the Province of Styria, and the Styrian Business Promotion Agency (SFG).

REFERENCES

[1] ISO 14229:2006(E). 2006. ISO 14229:2006: Road vehicles – Unified diagnostic services (UDS) – Specification and requirements. (2006).
[2] M. Cesana, L. Fratta, M. Gerla, E. Giordano, and G. Pau. 2010. C-VeT the UCLA Campus Vehicular Testbed: Integration of VANET and Mesh Networks. In *Proc. of the European Wireless Conference.*
[3] U. Drolia, Z. Wang, S. Vemuri, M. Behl, and R. Mangharam. 2011. AutoPlug – An automotive test-bed for Electronic Controller Unit Testing and Verification. In *Proc. of the IEEE Intelligent Transportation Systems Conference (ITSC).*
[4] N. Gabe. 2016. Over-the-Air Updates on Varied Paths. *Automotive News* (2016).
[5] I. Hossain, S.M. Mahmud, and M.H. Hwang. 2010. Performance Evaluation of Mobile Multicast Session Initialization Techniques for Remote SW Upload in Vehicle ECUs. In *Proc. of the IEEE Vehicular Technology Conference.*
[6] K. Koscher et al. 2010. Experimental Security Analysis of a Modern Automobile. In *Proc. of the IEEE Symposium on Security and Privacy.*
[7] M. Khurram et al. 2016. Enhancing Connected Car Adoption: Security and OTA Update Framework. In *Proc. of the 3rd World Forum on Internet of Things.*
[8] M. Steger et al. 2017. An Efficient and Secure Automotive Wireless Software Update Framework. *Under submission* (2017).
[9] M.S. Idrees et al. 2011. Secure Automotive On-board Protocols: A Case of Over-the-air Firmware Updates. (2011).
[10] D.K. Nilsson and U.E. Larson. 2008. Secure Firmware Updates Over the Air in Intelligent Vehicles. *IEEE Conference on Communications* (2008).
[11] R. Petri et al. 2016. Evaluation of Lightweight TPMs for Automotive SW Updates over the Air. In *Proc. of the Conference on Embedded Security in Cars.*
[12] Redbend Software. 2011. Updating Car ECUs Over-The-Air (FOTA). (2011).
[13] M. Steger, C.A. Boano, M. Karner, J. Hillebrand, W. Rom, and K. Roemer. 2016. SecUp: Secure and Efficient Wireless Software Updates for Vehicles. In *Proc. of the Conference on Digital System Design (DSD).*
[14] M. Steger, M. Karner, J. Hillebrand, W. Rom, C.A. Boano, and K. Roemer. 2016. Generic Framework Enabling Secure and Efficient Automotive Wireless SW Updates. In *Proc. of the Conf. on Emerging Technologies and Factory Automation.*
[15] I. Studnia, V. Nicomette, E. Alata, Y. Deswarte, M. Kaâniche, and Y. Laarouchi. 2013. Survey on Security Threats and Protection Mechanisms in Embedded Automotive Networks. In *Proc. of the Conf. on Dependable Systems and Networks.*
[16] W. Vandenberghe, I. Moerman, P. Demeester, and H. Cappelle. 2011. Suitability of the wireless testbed w-iLab.t for VANET research. In *Proc. of the 18th Symposium on Communications and Vehicular Technology in the Benelux.*
[17] T.L. Willke, P. Tientrakool, and N.F. Maxemchuk. 2009. A Survey of Inter-Vehicle Communication Protocols and their Applications. *IEEE Communications Surveys & Tutorials* 11, 2 (2009).

Paper G

M. Steger, C.A. Boano, T. Niedermayr, K. Römer, M. Karner, J. Hillebrand, and W. Rom.
An Efficient and Secure Automotive Wireless Software Update Framework. *To appear in IEEE Transactions on Industrial Informatics (TII)*, 2018.

Summary. In this paper, SecUp, a generic framework enabling wireless software updates and diagnostics is proposed. SecUp supports the whole life-cycle of a modern vehicle considering the requirements coming from different application scenarios and enables efficient as well as fast wireless software updates by allowing – besides providing basic automotive wireless software update functionality – beneficial features such as parallel software updates (where the same software binary will be installed on different vehicles and ECUs simultaneously) and partial software updates (where only changed parts of the software are transferred to the ECU). The paper first describes the role of involved devices and users in the considered application scenarios and provides an overview on the performed security analysis as well as the resulting security concept. Thereafter, the developed prototypes of the core nodes are presented, different automotive ECUs used to evaluate SecUp are described, and the developed features allowing parallel or partial software updates are explained. The evaluation of SecUp is performed in two steps: first, SecUp's security concept and its impact on the system performance is analyzed. Second, the evaluation of the proposed framework and the its advanced update mechanisms is described and the corresponding results are presented.

My contributions. As main author of this paper I wrote the vast majority of this paper supported by all co-authors, who supported the paper by numerous discussions regarding the structure of this paper as well as the different evaluation steps w.r.t security and efficiency. Thomas Niedermayr significantly supported the design of the bootloader for the AURIX ECU (i.e., our main demonstrator ECU) and implemented large parts of this bootloader. I carried out all presented case studies and developed all essential parts of the wireless update framework except of the demonstrator ECU.

©2017 IEEE.
ISSN: 1551-3203.
DOI: 10.1109/TII.2017.2776250.

Link: http://ieeexplore.ieee.org/document/8141933/.

An Efficient and Secure Automotive Wireless Software Update Framework

Marco Steger, Carlo Alberto Boano, *Member, IEEE,* Thomas Niedermayr, Michael Karner, Joachim Hillebrand, Kay Roemer, Werner Rom

***Abstract*—Future vehicles will be wirelessly connected to nearby vehicles, to the road infrastructure, and to the Internet, thereby becoming an integral part of the Internet of Things. New comfort features, safety functions, and a number of new vehicle-specific services will be integrated in future smart vehicles. These include a fast, secure, and reliable way to diagnose and reconfigure a vehicle, as well as the installation of new software on its integrated electronic control units. Such wireless software updates are beneficial for both automotive carmaker and customers, as they allow to securely enable new features on the vehicle and to fix software bugs by installing a new software version over the air. A secure and dependable wireless software update process is valuable in the entire lifetime of a modern vehicle as it can be used already during vehicle development and manufacturing process on the assembly line, as well as during vehicle maintenance in a service center. Additionally, future vehicles will allow to remotely download up-to-date software on the electronic control units.**

To support this process over the entire vehicle's lifetime, a generic framework is needed. In this paper, SecUp, a generic framework enabling secure and efficient wireless automotive software updates is proposed. SecUp utilizes IEEE 802.11s as wireless medium to interconnect vehicles and diagnostic devices in a dependable and fast way. Additionally, SecUp is enabling beneficial wireless software update features such as parallel and partial software updates to increase the efficiency, and comprises advanced security mechanisms to prevent abuse and attacks.

***Index Terms*—Automotive systems, Generic framework, IEEE 802.11s, SecUp, Security concept, Wireless software updates**

I. Introduction

MODERN vehicles include a growing number of electronic control units (ECU) in order to incorporate new services. The latter require elaborate and often distributed software (SW) implementations on the integrated ECUs of a vehicle, which increase the complexity of the SW. Furthermore, the growing connectivity and distribution of vehicular systems is potentially introducing a growing number of bugs in automotive SW implementations and associated functions.

Fixing such SW bugs as well as upgrading the ECU SW to enable new features requires new concepts allowing efficient automotive SW updates. Thereby, these concepts are supporting the development and maintenance of modern vehicles. Efficient and secure SW updates can be beneficial over the entire life-cycle of a modern vehicle and will significantly reduce the time needed for vehicle maintenance.

In this paper, SecUp, a generic framework enabling wireless SW updates and diagnostics is proposed. Contrary to existing works only focusing on wireless remote updates such as [1], [2], [3], [4], SecUp supports the entire life-cycle of a modern vehicle considering the requirements coming from different application scenarios (i.e., vehicle development, assembly, and maintenance in a service center). In all these scenarios, wireless SW updates performed locally in a dedicated area (e.g., the service center or assembly line) have to be fast, efficient, and secure. To enable efficient and fast wireless SW updates, SecUp supports – besides the basic automotive wireless SW update functionality – beneficial features such as parallel SW updates (where the same SW binary is installed on different vehicles and ECUs simultaneously) and partial SW updates (where only changed parts of the SW are installed).

Additionally, SecUp utilizes a comprehensive security concept based on strong authentication and encryption mechanisms to guarantee secure wireless SW updates. This security concept also ensures the integrity of the transferred data and of the entire vehicle. As a result, SecUp is protecting the diagnostic devices, the transferred data, the vehicles, and the OEM (Original Equipment Manufacturer, i.e., the vehicle manufacturer) backbone from unauthorized access.

The key contribution of this paper is the introduction of SecUp, a generic framework allowing efficient and secure wireless SW updates applicable for different automotive application scenarios, namely: (i) SW updates in the vehicle development, (ii) in the assembly line, as well as (iii) during maintenance in service centers. SecUp allows not only basic wireless SW updates, but additionally addresses the efficiency aspect by enabling parallel and partial SW updates. Furthermore, SecUp is built upon a novel security concept.

This paper is structured as follows. After reviewing related work in Section II, SecUp and its architecture are described in Section III. Additionally, the advantages of the employed IEEE 802.11s network are listed, proving its applicability to perform efficient and reliable wireless SW updates. In Section IV the roles of involved devices and users in the considered application scenarios is highlighted. Section V provides an overview on the performed security analysis as well as the resulting security concept. In Section VI, the developed prototypes of the core nodes are presented and different automotive ECUs used to evaluate SecUp are described. At the same time the developed features allowing parallel or partial SW updates are presented. To evaluate SecUp, first, in Section VII, SecUp's security concept and its impact on the system performance is analyzed. Second, the evaluation of SecUp is described and the corresponding results are presented in Section VIII. Finally, a conclusion is given in Section IX.

M. Steger, M. Karner, J. Hillebrand, and W. Rom are working at the VIRTUAL VEHICLE research center.

C. Boano, T. Niedermayr, and K. Roemer are working at the Institute for Technical Informatics, Graz University of Technology, Austria.

II. Related work

The benefits of automotive over-the-air (OTA) updates compared to traditional (i.e., wired) SW updates are listed in [1] and a high-level architecture for these updates is presented. However, a description of the wireless medium, required security mechanisms or other technical details are not included. The authors of [5] described the SW update process for ECUs based on international standards. However, this work is not addressing a wireless approach for such updates at all.

Other authors of previous works such as [2], [3], [4], [6], [7], [8] only focus on remote SW updates for vehicles and the corresponding security issues but neither consider other scenarios where updates are performed locally (e.g., within a service center), nor allow advanced SW update mechanisms such as parallel SW updates. A system allowing OTA updates was presented by Idrees et al. [2]. This system utilizes a Hardware Security Module (HSM) for data encryption, key management as well as to ensure data integrity on both the wireless interface and all ECUs of a vehicle. Therefore, a HSM is required on every ECU, which leads to significant extra costs. Additionally, the authors do not give any insight regarding the specific type or the properties of the wireless link. The authors of [9] also propose to use security HW modules on the ECUs and the vehicle gateway to secure OTA updates. Thereby, the authors focus on how to integrate these modules in the ECUs and evaluate the resulting resource overhead. The paper also contains a high-level description of an architecture for OTA updates, however, no implementations details or evaluation results are given. Nilsson et al. [3], [4] propose a system allowing automotive OTA updates by connecting a vehicle to a portal server using an Internet link. Thereby the authors list important security aspects, especially data integrity and data confidentiality, w.r.t. OTA updates but neither describe the utilized wireless network nor address the data flow in the network. In [7] an architecture for secure wireless SW updates sending multiple copies of the SW to ensure data integrity is presented. However, this solution is only addressing point-to-point links between one vehicle and an OEM server and relies on numerous prerequisites (e.g., ensuring data integrity of OTA updates by sending multiple copies). The authors of [8] also propose to send a SW binary two times to a vehicle to secure the update process without providing a detailed description on the actual SW update framework or other technical insights.

Although previous works are proposing different systems allowing remote OTA updates and addressing the corresponding security aspects, the listed solutions are only addressing unicast links (i.e., a point-to-point connection between the OEM and a specific vehicle) and hence, novel features such as parallel or partial SW updates cannot be realized. Contrary to these works, SecUp allows to install the latest SW on several vehicles simultaneously and provides efficient SW update mechanisms (i.e., parallel or partial SW updates).

To date (2017), Tesla is the only car manufacturer providing a solution for automotive OTA updates. A wireless network (either based on 3G/4G or a Wi-Fi network connecting the vehicle to the Internet) is utilized to transfer the latest SW from the servers of the OEM to a Tesla vehicle [10]. However, this point-to-point connection between the OEM and the vehicle cannot be used to simultaneously install SW in different vehicles and on several ECUs in parallel.

The authors of [2], [3], [4], [7] were mainly focusing on different security aspects of automotive SW updates and have listed a number of relevant security threats, namely: *data confidentiality*, *data integrity*, *key exchange and management*, and *vehicle integrity and authentication*. Different from previous work, the security concept proposed in this paper and presented in Section V-B addresses all these aspects at once.

A. IEEE 802.11s mesh networking and its security features

The proposed framework for wireless SW updates proposed utilizes a IEEE 802.11s network to interconnect vehicles and diagnostic HW. Today, it is the only solution using 11s as medium for wireless SW updates, however, the protocol is utilized in other automotive applications: the authors of [11] and [12] are using IEEE 802.11s as a backbone network for vehicle-to-vehicle and vehicle-to-infrastructure communication (V2X) networks to interconnect V2X entities and road side units. In [13] different wireless communication protocols such as IEEE 802.11 (Wi-Fi), Bluetooth Low Energy (BLE), IEEE 802.15.4 (ZigBee) and IEEE 802.11s are compared with each to find the most suitable protocol for wireless SW updates in an automotive environment. Thereby, the authors focus on different aspects such as throughput, scalability as well as extendability, show that IEEE 802.11s is the only protocol able to satisfy all investigated aspects, and point out weaknesses of other protocols (e.g., BLE and ZigBee offering insufficient throughput). Although the authors reveal the applicability of IEEE 802.11s for wireless automotive SW updates, the paper does not include any descriptions of a framework enabling these kind of updates. The benefits of utilizing an IEEE 802.11s network to perform wireless automotive SW updates are listed in Section III-B.

Several contributions mainly focusing on the security aspects of IEEE 802.11s have proposed different extensions and improvements. However, these aspects were not discussed w.r.t. automotive applications. Tan et al. [14] describe internal as well as external attacks on IEEE 802.11s networks and list relevant security requirements. The authors state that Simultaneous Authentication of Equals (SAE) [15] used in IEEE 802.11s is able to mitigate external attacks. However, a range of internal attacks is not prevented by SAE and therefore additional security features are required.

In [16] GPS positioning is used to mitigate a wide range of potential attacks on IEEE 802.11s. The proposed system called PASER is compared to other approaches for secure IEEE 802.11s networks in [17], and the authors show, beside others, the inefficiency of these approaches w.r.t. time and power. The use of GPS, as proposed in [16] and [17], however, is not applicable within the assembly line or a service center and therefore PASER cannot be utilized by SecUp.

The approaches proposed in [14], [18], [19], [16], [17] allow the creation of secure IEEE 802.11s networks, however, none of these approaches is able to fulfill both the efficiency requirements of wireless SW updates and the immunity against possible attacks. The defined security concept presented in Section V is fulfilling both aspects by utilizing security mechanism on the network as well as the application layer. In particular, the proposed security concept is based on a structured security analysis. In [20] the latter was described in detail. This paper also showed how the analysis can be utilized to design automotive applications w.r.t. SAE J3016 [21], the new security standard in the automotive domain.

III. Framework and Architecture

This section describes the architecture of SecUp. A system overview is given in Section III-A and the utilized wireless network is addressed in Section III-B.

A. System Architecture and Core Nodes

Secure and efficient wireless SW updates will, independently from the application scenario, require a reliable and fast wireless network. Furthermore, a dependable smart gateway interface is needed for each vehicle to interconnect the latter with the wireless network in a reliable and secure way. This interface is the most critical component of SecUp and is called Wireless Vehicle Interface (WVI). A WVI interconnects the vehicular communication infrastructure (i.e., automotive bus systems such as Controller Area Network (CAN)) and the ECUs of a vehicle with a wireless network and in further consequence with the Internet.

A WVI can be realized either as a fully integrated device (i.e., a smart bus gateway or a dedicated ECU) or as a plug-in solution. The latter can be temporarily connected to a vehicle using its On-Board Diagnostics (OBD) interface and is mainly utilized in service center scenarios, where the plug-in property of the WVI and the utilization of standardized in-vehicle communication protocols are ensuring backward compatibility as well as OEM-independence. In future vehicles the WVI will be part of the in-vehicle communication system, thereby enabling new services and functions which require to access data generated by or stored in the vehicle.

The so-called Diagnostic Tester (DT) can be seen as source of new SW versions within the wireless SW update framework. It can use a backbone link to the OEM to obtain the latest available SW as well as required information about the vehicle such as the vehicle configuration (e.g., types and IDs of the ECUs) or required authorization keys. Depending on the scenario and other general conditions, a DT can either be a dedicated device (e.g., tool in a service center) or more likely a SW application running on a laptop, PC, or server. Additionally, a DT typically supports various diagnostic functions. In typical OTA update scenarios, the DT application is running on a server of the car manufacturer and a secure Internet link (e.g., a VPN tunnel) between the WVI and the DT is established.

In SecUp, hand-held devices connected to the wireless SW update system are used by mechanics in a service center or by engineers during the vehicle development phase i) to run wireless diagnostics, ii) to monitor messages exchanged on the different bus systems of a vehicle, and iii) to trigger, monitor, and validate the SW update process.

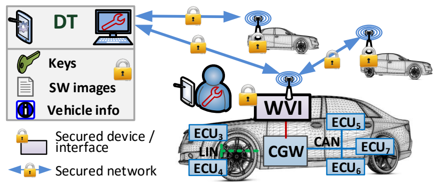


Fig. 1. An IEEE 802.11s network applied in a typical service center scenario. Mechanics use handhelds to run wireless diagnostics or wireless SW updates.

B. Wireless IEEE 802.11s network

To interconnect all involved nodes described in Section III-A, SecUp utilizes an IEEE 802.11s network. According to [13], this protocol is most suitable for wireless automotive SW updates and outperforms other wireless standards such as BLE or ZigBee in different important aspects like throughput, scalability as well as extendability. IEEE 802.11s is based on a mesh network topology, where each node will either directly communicate with other nodes in its transmission range or use other nodes in between to forward a data packet to the intended destination. The mesh characteristics of IEEE 802.11s allow a data packet to take different paths when sent through the network. This implicit redundancy increases the reliability of IEEE 802.11s networks. Similarly, the transmission range of a network can be increased by adding relay nodes (e.g., other vehicles or devices) at the edge of the network.

The multihop capability of IEEE 802.11s is a key advantage that significantly increases the reliability and the availability of such a mesh network, and this allows the use of an IEEE 802.11s network also in harsh radio environments.

A modern service center, as sketched in Figure 1 and further described in Section IV-B, is a typical example of a harsh environment, as wireless links can be affected by the shielding of vehicles or other (metal) objects, which potentially decreases the reliability of data transmission / collection. An IEEE 802.11s network, however, is able to provide a stable communication: if a direct link between the vehicle and the DT is too weak to exchange a packet, other vehicles or IEEE 802.11s relay nodes located in between will forward the data packets to the target vehicle or the DT.

The IEEE 802.11s standard includes multicast data streams in mesh networks. Such a multicast can be used to send data packets from one node to several other nodes in a network. SecUp can hence potentially use such layer two multicasts to transfer and install a new SW version on several vehicles simultaneously (i.e., to carry out parallel SW updates). Although the IEEE 802.11s standard defines multicast data streams, the current open11s implementation [22] used in the developed framework does not support multicasts on layer two yet and therefore multicast is implemented on higher layers.

IV. Supporting the Entire Vehicle Lifetime

In this section, the considered scenarios for wireless SW updates are listed first. Thereafter, an illustration of wireless SW updates in a service center is given in Section IV-B.

A. *Wireless software update scenarios*

SecUp can satisfy the requirements of wireless SW updates in the following scenarios: i) during vehicle development, ii) in the assembly line, and iii) in a service center. In the following, the scenario-specific requirements are highlighted, and information about involved users, available infrastructure, and related security concerns are given.

1) Vehicle development: during the vehicle development phase, engineers will have to update the SW of one or more ECUs of a test vehicle several times to analyse, compare, and evaluate newly developed features. Therefore, the development engineers require a flexible and efficient system enabling wireless SW updates as well as vehicle diagnostics. Vehicle development activities will mainly take place in restricted areas and will be performed by engineers (i.e., expert users). The wireless solution offers several advantages in this scenario, as it enables fast and flexible SW updates while allowing the usage of hand-held devices like tablets or smartphones.

2) Vehicle assembly: this step is performed in a highly automated and secure environment where many operations are performed by machines and robots. Before a vehicle can leave the assembly line, the latest SW is installed on all integrated ECUs of a vehicle. Therefore, the SW of many vehicles must be updated – ideally in parallel – to install the latest SW on all ECUs. Because of the high number of vehicles as well as the high degree of automation, scalability, reliability and efficiency of the SW update system are very important.

3) Vehicle maintenance: in a service center mechanics will diagnose, repair, and maintain several vehicles. Therefore, a mechanic will connect to a vehicle to run diagnostic functions, to check if there are any Diagnostic Trouble Codes (DTC) sent by the vehicle and its ECUs, and to perform the necessary repairs. Thereby, the mechanic will also check if new SW is available for one or several ECUs of the vehicle and install it. Simultaneous wireless SW updates would be very beneficial especially if large vehicle recalls (e.g., due to a critical software bug) are necessary: a mechanic can connect to several vehicles in parallel and install the new software simultaneously. Additionally, a wireless solution can be very beneficial in service centers as a mechanic will not have to use heavy diagnostic equipment (e.g., a PC and a battery contained in a solid metal case), but can run wireless diagnostics and SW updates utilizing a lightweight hand-held device instead.

4) Remote SW updates: mainly relevant for future vehicles with an integrated WVI, either realized as bus gateway or as dedicated ECU, allowing wireless connectivity via 3G/4G or Wi-Fi. The current version of SecUp is mainly focusing on SW updates performed in local environments, however, the developed SW updates mechanisms could also be used when connecting the vehicle to the wireless network of the user.

Scenario-specific particularities are summarized in the following: SW updates in the assembly line or during the vehicle development phase are performed by expert users in secured and restricted areas. In these use cases security is an important issue (e.g., industrial espionage) but not as critical as in OTA scenarios, where the update is performed by an untrained user at the users' home or in public, potentially utilizing a compromised device or an insecure network. Especially the service center and the assembly line scenarios require a very efficient and fast way to install SW updates. During vehicle development high flexibility by easily extending the transmission range of the wireless network is especially required due to the big variety of function tests, system evaluations, and diagnostics performed during this phase.

B. *Wireless software updates in a service center*

In this section the required steps to install latest SW on a vehicle's ECU are described by utilizing a typical service center scenario as shown in Figure 1. In this sketched example, mechanics maintain vehicles by using handhelds to run wireless diagnostics and to perform wireless SW updates. Similar procedures and schemes apply in the other update scenarios. The main goal of this section is to explain the basic SW update procedure without providing details on the involved security mechanisms, as this is described in Section V-B.

1) Connecting the vehicle to the SW update system: a mechanic first connects a WVI to the vehicle (i.e., if the vehicle doesn't have an integrated WVI) using its OBD interface. The WVI joins the IEEE 802.11s network and will connect to a DT once a periodically broadcasted beacon advertising the presence of the DT was received. The same principle is used by the handhelds to connect to the DT (i.e., after receiving a DT beacon) as well as to WVIs (i.e, by sending beacons). It is important to note that in case of an update, the SW is directly transferred from the DT to the WVI and is never stored on the handheld devices due to security reasons.

2) Gathering information about the vehicle: once the connections between the DT, the handheld device and the WVI are established, the DT starts to gather information about the vehicle by retrieving the Vehicle Identification Number (VIN). With the retrieved VIN, the DT can either query its local database or access an OEM server to obtain vehicle-specifics including vehicle model and variants, the integrated ECUs including ECU-IDs and the CAN-IDs, as well as available SW versions. This information is often condensed in so-called Open Diagnostic data eXchange (ODX) [23] files.

3) Performing the wireless SW update: if new SW for a vehicle is available, the SW binary will first be fully transferred from the DT to the WVI. The latter will then verify the received SW binary and start to install the binary on the ECU utilizing the Unified Diagnostic Service (UDS) protocol [24]. Besides this basic SW update mechanism, SecUp also supports *partial SW downloads*, where only the difference between the current and the new SW version is sent to the ECU. This feature can significantly reduce the duration of a SW update, however, it has to be supported by the ECU (see also Section VI-D3). To perform a SW update on several vehicles in parallel, a mechanic will first register vehicles for a certain SW update on the DT, which will then automatically start the update once all vehicles are ready to receive the SW. Parallel SW updates can again significantly shorten the duration of wireless SW updates and thereby reduce costs as several vehicles are addressed simultaneously.

V. Security and trust

Security is a critical aspect of wireless SW updates, as an attacker can compromise involved devices and protocols to reveal sensitive data and keys. Depending on the application scenario, different attack vectors have to be considered and therefore different levels of security are required.

SecUp's security concept is built on a system-centric design employing a measurable security approach as described in [25]. The used design approach, the DEWI[1] security metric, is based on a structured system decomposition rather then a traditional approach where first attack vectors are analyzed and second corresponding countermeasures are implemented.

Such a system-centric design encompasses the following essential steps: i) **security analysis** of the framework resulting in a secure system configuration, ii) **extraction of the security requirements** from the secure system configuration, iii) **Security concept definition** based on these requirements and the peculiarities of the application scenario, and iv) **evaluation of the defined security concept** using the STRIDE threat model [26]. [20] provides more detailed information and also highlights that this approach can be aligned with the new SAE security standard SAE-J3061 [21].

In the following the most important revealed security threats are stated and thereafter, in Section V-B, the security concept is described in more detail. This description also shows how the SecUp's security concept is able to mitigate these threats. More detailed information can be found in [27].

A. Security threats and attack vectors

A system allowing wireless automotive SW updates is a worthwhile target for a wide range of attacks. In the following we will collect and describe critical external as well as internal threats and show in a later step that the security mechanisms employed by SecUp are able to prevent these threats.

An external attacker can try to eavesdrop the wireless channel to reveal transferred authorization keys and the latest SW itself, or tamper with the transferred data to install malicious SW on the ECU of a vehicle (Threat T1).

Besides the aforementioned external attacks (e.g., committed by a hacker without access to the wireless network), there are also a range of internal threats performed by insiders such as a rogue mechanic in a service center or other users fraudulently using the wireless network. These insider threats encompass theft of equipment such as a WVI to perform unauthorized SW updates (e.g., tuning; T2), to extract secret keys stored on the device (T3), or spoofing the identity of an WVI or a DT (T4) to gather keys, the SW, or vehicle access.

To mitigate all (aforementioned) threats, SW update frameworks like SecUp must i) employ strong authentication schemes to establish trust between the involved devices, ii) use strong encryption as well as message authentication codes (MAC) to protect confidentiality as well as the integrity of the transferred data, iii) securely store sensitive data and secret keys in dedicated memory or HW security modules (HSM).

B. Security concept

The security concept presented in this section is built upon the results of the system analysis utilizing the DEWI security

[1]EU project Dependable Embedded Wireless Infrastructure (DEWI).

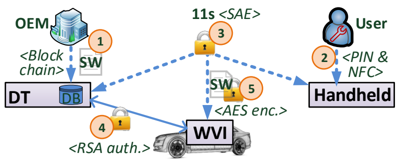


Fig. 2. Security architecture and flow. New SW is first securely distributed by the OEM [28] and stored on the DT. To secure the local update 2) a user authentication is performed, 3) a secured wireless network is established, 4) strong authentication is used between the core nodes, and 5) symmetric keys are used to securely transfer the new SW to the WVI.

metrics. The resulting security concept is based on security mechanisms on the network as well as on the application layer.

On the network layer, *wpa_supplicant*, a generic security framework for different types of wireless networks, is utilized to secure the IEEE 802.11s mesh network. The *wpa_supplicant* is part of the current open11s implementation and allows to secure the network in a lightweight way by using SAE [15]. SAE was developed especially for 802.11s-based, multihop capable mesh networks, is fully integrated in the latest *wpa_supplicant* version, and is able to mitigate a wide range of external attacks [14]. However, SAE is not providing suitable countermeasures against internal attacks, where attackers use nodes already connected to the wireless network (e.g., a compromised device or a rogue user such as a mechanic in a service center; threats T2-T4) to launch an attack. Therefore, additional security features are employed on the application layer to prevent internal attacks and thereby to address the four important security aspects, namely, vehicle integrity and authentication, data integrity, data confidentiality, key management and exchange.

To ensure the integrity of a vehicle, especially when equipped with a WVI, strong authentication mechanisms must be applied to keep unauthorized users from accessing the vehicular bus system. Additionally, it is even more important to avoid that an attacker endangers a whole fleet of vehicles by breaking one vehicle and extracting a shared (i.e., symmetric) authentication key. The developed concept is based on *unique asymmetric key pairs* consisting of a private and a public key used on the WVIs, handhelds, as well as DTs to ensure an unambiguous authentication between all nodes.

In a classic Public Key Infrastructure (PKI) the public keys are exchanged (e.g., over the Internet) and then used to encrypt data or to verify digital signatures. To allow a user to check if a public key really belongs to a certain entity, classic PKI systems employ a third party, the Certificate Authority (CA). As a consequence, each node requires an Internet connection to communicate with a CA.

In SecUp, a different approach has been chosen: to keep the system local (i.e., only the DT is connected to the Internet and/or the OEM backbone), a security concept without a CA was designed. However, such a concept requires that public keys are initially exchanged and then securely stored. This pairing step is performed in a controlled environment (e.g., close proximity to the DT) using a dedicated media or mechanism (e.g., SecUp supports NFC and the use of one-time passwords) by authorized users (e.g., head of a service center).

The pairing only has to be done once: first, a *master key pair* is created on each device and securely stored in dedicated memory or in a Trusted Platform Module (TPM). Second, the public keys are exchanged and then securely stored. After the initial pairing step, the master keys can be used to handle the authentication between the involved nodes and, additionally, to sign as well as to encrypt unicast packets.

As a WVI securely stores the public key of the DT (and vice versa), digital RSA (cryptosystem by Rivest, Shamir und Adleman) signatures (RSA signature with 2048bit key length and SHA-256 hash) can be used to authenticate the involved parties. In the authentication step (e.g., performed on a daily basis), the entities agree on a symmetric session key[2] which is used to ensure confidentiality and integrity of the data transfer. Please note that in SecUp, i) data packets are protected using session keys and not by utilizing the master keys as the use of symmetric keys is more efficient, and ii) timestamps and nonce are employed to protect exchanged messages and thus to mitigate common attacks such as replay attacks, similar to TLS and other common secure message transmission protocols.

Secure multicast data streams required to allow parallel SW updates, however, need symmetric keys to verify and encrypt data. Data sent from a DT to several vehicles must be signed and encrypted using a shared key. This shared *session key* is created by the DT and then distributed individually to all WVIs using a unicast packet signed with the private master key of the DT. In further consequence, multicast data packets are encrypted and signed using the *session key* to ensure the confidentiality and the integrity of the exchanged data.

By utilizing the *master key pairs* and the (shared) symmetric *session key*, the security aspects authentication, confidentiality, and integrity can be solved w.r.t. the vehicle itself, as well as on data exchange level (unicast as well as multicast). To prevent an attacker from extracting the utilized keys (e.g., by stealing a WVI), these keys must be stored securely on the devices and kept secret all the time. Therefore, keys used in SecUp are stored in dedicated secure memory or TPMs.

C. Application of the security concept to a use case

The service center scenario sketched in Section IV-B will again be used to illustrate how to secure wireless SW updates by applying the defined security concept (see also Figure 2):

1) User authentication: mechanics authenticate with the system using a NFC smartcard and a PIN code. Different user profiles are used to authorize different modes: normal mechanics can use SecUp to run wireless diagnostics only. A privileged user can additionally perform wireless SW updates.

2) Interconnecting the WVI: after connecting the WVI to a vehicle, the WVI powers up and connects to the IEEE 802.11s network using a shared network key.

3) Authentication between WVI and DT: the master keys of the WVI and the DT are used to authenticate with each other.

4) Parallel SW updates: first, the DT creates a session key and distributes it to all concerned WVIs (unicast). Second, the DT sends the fragmented SW binary to the WVIs by signing and encrypting every packet using the session key (multicast).

[2]Advanced Encryption Standard (AES) in Galois/Counter Mode (GCM) – AES-GCM

5) Data verification: to ensure data integrity, the DT first computes the hash value of the SW binary. Then the DT signs and encrypts the hash value using the master keys and sends it to the concerned WVIs (unicast). Hence, these WVIs use the transferred hash to verify the received SW binary before installing it on the concerned ECUs. Please note that the genuineness of new SW (i.e., that SW is really coming from the OEM) is verified by the DT before a new SW binary is stored in the local database of the DT. However, this secure SW distribution process (see [28]) is out of the scope of this paper. In the current concept, the WVI is not verifying the genuineness again, as the DT already verified it.

D. Formal security concept evaluation

The defined security concept was first formally evaluated using the Microsoft STRIDE threat model [26]. STRIDE is an attack-centric approach that can be used to analyze the security of a system by identifying a number of potential security threats and grouping them into six categories: **S**poofing, **T**ampering, **R**epudiation, **I**nformation disclosure, **D**enial of service, and **E**levation of privilege. To apply STRIDE, each part of the system is analyzed and all potential threats for every component or process are determined. Based on these threats, suitable countermeasures can be defined in a subsequent step. SecUp's security concept evaluation was performed in a similar way: first a list of potential STRIDE threats was created, and a prioritization of these threats was performed w.r.t. likeliness and severity. Second, these threats were (theoretically) applied to the security concept and its security features to prove that suitable countermeasures exist to avoid all of these threats. In [27] the evaluation process is described in more detail and the results of the formal analysis are presented.

Due to space constraints a complete threat analysis and a system evaluation w.r.t these identified threats is not possible. However, in the following the threats T1-T4 described in Section V-A are used to show the security features of SecUp. Threat T1 is addressing confidentiality and integrity of exchanged data. In SecUp, exchanged data is secured by i) SAE on network level, used to protect the IEEE 802.11s network and mitigating external attacks, and ii) security features on the application layer encompassing per-packet encryption and integrity checks as well as a final verification of the transferred SW binary by sending the hash value of the binary (encrypted and signed by the DT) to the WVI.

Threats T2 and T3 are addressing theft of equipment (e.g., a plug-in WVI or a handheld in a service center). SecUp allows wireless SW updates within dedicated areas like a service center: after an initial pairing step between the DT and a new WVI, the used equipment is forming a trusted network and SW updates will only be allowed when a WVI is connected to a trusted DT. A stolen WVI can therefore not be used to perform unauthorized SW updates (T2), as the WVI will not allow an attacker to perform an update without being connected to the trusted DT. The attacker can in a second step try to spoof the identity of the trusted DT (T4), however, due to the employed authentication step using digital signatures, also this attempt will fail. An attacker can also use stolen equipment to read or replace keys stored on the device, however, will fail as SecUp utilizes TPMs to securely store keys and other sensitive data.

VI. Implemented prototypes and demonstrators

In this section the implemented HW and SW prototypes of the core nodes of the wireless SW update system are described and the utilized demonstrator ECUs are presented.

A. The developed WVI prototype

The developed WVI prototype consists of a BeagleBone Black (BBB) board and a developed Printed Circuit Board (PCB). This PCB can be mounted on the BBB using the designated header pins and encompasses the required HW interfaces (i.e., CAN and OBD) as well as the corresponding transceivers to interconnect a WVI and a vehicle. The PCB is also responsible for the battery management of a WVI: the battery is used as power supply of the WVI when the ignition of the vehicle is off and no power is provided via the OBD interface of the vehicle. The SW implementation of the WVI is mainly done in Java and the Java Native Interface (JNI) is used to interact with the HW-related parts of the WVI prototype.

A key aspect of the developed security concept is to use strong authentication mechanisms between the involved devices (i.e., WVIs, handhelds, and DTs). The corresponding authentication keys must be kept secret to guarantee a trustworthy system. Especially the plug-in WVI is critical, as an attacker can steal such a device and try to extract the secret keys. To avoid that, a TPM can be used on the WVI to securely store secret keys and other sensitive material. Although, the TPM is currently only used to store RSA keys and to handle the authentication between the WVI and the DT, it would also allow the use of certificates in future versions of SecUp.

1) Trusted platform module integration: The used TPM from NXP is connected to the WVI via Inter-Integrated Circuit (I^2C) bus. The I^2C protocol required to exchange data between the TPM and the WVI was not fully supported by the I^2C library available on the BBB. Because of that, an additional I^2C bus using normal I/O pins was implemented on the BBB to fulfill the requirements of the TPM w.r.t. the I^2C communication protocol. The disadvantage of this approach, however, is that the communication between the TPM and the BBB is rather slow: about 7 kBaud.

In Section VII-A the impact of the slow bus connection between the WVI and the TPM is evaluated by first comparing the performance of the integrated TPM with the software-based cryptography library Java Bouncy Castle (JBC) and second using an oscilloscope to analyze the timing behavior of the TPM when performing different cryptographic operations.

B. Prototypes and implementation of the DT and handhelds

The DT implementation was developed in Java and tested on both a dual-core laptop running Win7 as well as on a BBB running Debian Linux (the system evaluation was performed by an BBB running as DT). The latest DT implementation supports different modes such as ECU programming, monitoring of the CAN bus, and OBD diagnostics.

A Nexus 7 Android tablet is used as hand-held device. It can be connected to a WVI and a DT simultaneously, thereby empowering the hand-held device to monitor the bus systems of the vehicle, to run OBD requests, and to trigger SW updates.

C. Volvo FlexECU used for basic framework evaluation

The Volvo FlexECU is a prototyping ECU mainly used to test new applications. The ECU offers several connectors (i.e., CAN interface, power supply, and enable pin) and comes inside a solid metal case. The bootloader of the ECU contains an UDS stack and thus supports a UDS-compliant SW update process. A reset must be triggered to access the bootloader when the ECU is already running its application SW.

This ECU was used to perform basic evaluations of the SW update framework (Section VIII). Two slightly different SW versions, one periodically sending CAN frames with ID 1001 and the other with ID 2001, were used to test the developed framework and to run some basic performance analysis. Due to the different CAN-IDs it is easy to validate that a new SW version was correctly installed on the ECU.

D. Infineon AURIX as advanced ECU demonstrator

The Volvo ECU is not suitable to test advanced features such as delta or partial wireless SW updates as neither the FlexECU HW nor the SW running on the ECU can be adapted or extended. Because of that, a second demonstrator ECU based on an Infineon AURIX was developed. Utilizing AURIX brings several advantages and offers more freedom to develop advanced SW update features, as the default bootloader can be modified or even a new AURIX bootloader can be developed.

1) AURIX platform description: the developed demonstrator ECU consists of an Infineon AURIX multi-core ECU integrated in the AURIX application kit TC277 TFT. It is a high performance ECU compliant to support safety requirements up to ASIL-D[3], the highest automotive safety level [29]. The AURIX ECU is based on a 32 bit scalar TriCore CPU running at up to 300 MHz in the full automotive temperature range $\{-40, +170\}$°C. It is equipped with up to 4MB flash and 472KB RAM memory and comes with high speed CAN transceivers, a safety processor and watchdog, as well as dedicated closely-coupled memory areas per core.

2) Flash-over-CAN mechanism for AURIX: this mechanism is the basis for wireless SW updates as it encompasses the transfer of new SW from the WVI to the AURIX ECU utilizing the CAN bus as well as the installation of this SW on the ECU. AURIX does not provide such a mechanism by default. Therefore an extended AURIX bootloader enabling a reliable flash-over-CAN mechanism was developed. The latter basically consists of three main components: i) a *CAN driver* handling the data transfer over CAN, ii) a *flash driver* required to program and erase the flash memory, and a iii) *controller* coordinating the programming sequence.

The CAN driver itself is already implemented in the AURIX basic SW and must only be configured to support the right bitrate (e.g., 1Mbit/s) of the bus. The flash driver is responsible

[3]Automotive Safety Integrity Levels (ASIL) defined from A to D

for programming and erasing the data as well as the program flash memory of the ECU and basically consists of two main functions: erasing of sections and programming of pages. The flash driver is able to erase one or more consecutive sections and can write single pages or use the burst programming mode to write multiple consecutive pages of the flash memory. The controller component was developed compliant to the UDS standard and is started each time the AURIX bootloader is called (e.g., after a HW or SW reset). It initializes the CAN driver and checks for programming requests. If no such request was received, it starts the ECU's application SW or reboots if no application SW is present. Otherwise, if a programming request was found, the controller handles the data transfer as well as the installation of the new application SW.

New application SW can be installed on AURIX by first sending a programming request containing the start address of the SW to the ECU. Second, an authorization step based on a typical Seed & Key mechanism is performed and then an UDS-compliant programming session is started on the ECU. In the third step, the SW is transferred to the ECU block-by-block using the corresponding UDS commands. The ECU receives a block, stores it in a temporary buffer and finally utilizes the flash driver to write it to the flash memory. After transferring all blocks to the ECU, the new SW version is validated by computing the Cyclic Redundancy Check (CRC) over the new SW blocks. Finally, the ECU reboots and starts the new application.

3) Partial SW updates: transferring the SW binary to the ECU via CAN bus is, according to Section VII-C and the results shown in Table I, by far the most time consuming step when performing a wireless SW update. Speeding up this step will significantly decrease the duration of a SW update.

One way to achieve this is to utilize partial SW updates. The basic idea behind partial SW updates is to only update the parts of the software that have changed compared to the currently installed SW version. The possible benefit can be up to nearly 100% in case of a simple parameter value change: a normal SW update will require to download the entire new binary to the ECU regardless of whether the old and the new binary are very similar (i.e., just one or a few parameters have changed) or if the new version has a lot of new features and is therefore much bigger in size. Partial SW updates will, in contrast, only download the changed parts of the binary (i.e., one or several memory blocks). In flash architectures, the available memory is divided into different sections, sometimes even with different sizes. To manipulate a section, a flash driver first has to erase the entire flash section before it can be filled with new content. Therefore, it is not possible to just update the value of one parameter stored in a specific section, but the entire memory section has to be rewritten. For partial SW updates this fact leads to two different approaches:

Delta download: only relevant (i.e., the changed) parts of a SW section are sent to the ECU. On the ECU the affected section is copied into RAM memory, the flash section is erased by the flash driver, the content of the section (in RAM) is modified using the received delta bytes, and finally the resulting section content is written to the flash memory again. This approach on the one hand minimizes the amount of data to be transferred to the ECU, but on the other hand significantly increases the complexity at ECU level.

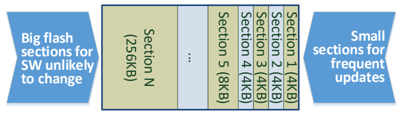


Fig. 3. Flash memory is divided into sections with different block sizes.

Partial SW update: instead of only transferring delta bytes, the entire section is transferred to the ECU. Although, this limits the benefit of a partial SW update as more data must be sent to the ECU, it is very simple to implement.

To decide whether to use delta downloads or to utilize partial SW updates, different factors such as section size, bitrate of the bus, as well as the bus load must be taken into account.

For AURIX ECUs the partial SW update approach was chosen as its flash memory is divided into sections with different sizes (see Figure 3) and as the SW developer can utilize a linker file to influence where specific parts of the SW binary are stored in the flash memory. When developing new AURIX SW, the binary shall be divided into different sections and mapped to the ECU flash memory accordingly as illustrated in Figure 3: areas that are most likely unchanged in the vehicle's lifetime (e.g., basic ECU functionality) and areas that are subject to change (e.g., blocks containing parameters or optional features). The latter will be located in one or several small sections (mostly a few KBs) of the flash memory.

VII. Evaluation of the Security Concept

In this section the impact of the employed security mechanisms on the system performance is evaluated.

A. Evaluating the performance of the integrated TPM

In SecUp a TPM is integrated in the WVI to securely store secret keys and sensitive data. The used TPM is capable to store keys and to perform cryptographic operations such as RSA encryption/decryption or RSA/AES key creation. The TPM can also be used for symmetric data encryption and signing. Therefore all cryptographic tasks required by the defined security concept are supported by the TPM. However, they can also be performed by the used Java SW-library JBC.

In a first evaluation step the performance of the TPM and the JBC is compared. Therefore, the authentication step between a WVI and a DT is performed 20 times first using the TPM and second utilizing JBC and thereby the duration of these operations was measured. The authentication step including different RSA operations (i.e., data encryption/decryption and signing) can be performed by the TPM, the JBC library, or as hybrid solution utilizing both in a combined way.

The gathered results show that the JBC outperforms the TPM: min/max/avg duration when using the TPM (6328/6588/6415.5ms) and JBC (575/701/635.8ms), respectively. However, the most important advantage of the TPM is its ability to securely store sensitive data and keys. Because of that, the final concept uses a hybrid solution (3525/3690/3577.9ms) where the fast JBC library is combined with the secure storage of the TPM. As the performance difference between JBC and the TPM, a HW chip dedicated to perform cryptographic operations, is not really obvious in the first place, an additional evaluation was performed to get to the bottom of this issue.

As mentioned in Section VI-A1, the I^2C bus connecting the TPM with the WVI was realized using the I/O pins of the BBB and therefore the speed of the bus is rather slow (7 kBaud). To evaluate the impact of the slow bus connection on the overall performance of the TPM, several RSA operations (RSA encryption/decryption with 2048 bit key, 208 byte data) were performed and meanwhile the timing behavior of the TPM was analyzed using an oscilloscope. The measurements clearly show that the I^2C bus has an significant impact on the overall performance as the time required for the data exchange between the TPM and the WVI is about 45% for RSA decryption and up to 87% for RSA encryption of the overall duration of the operation. The measurements reveal that either a faster I^2C bus or a different bus such as Serial Peripheral Interface (SPI) would help to significantly decrease the overall duration. In the current setup, the TPM shall only be used to store keys and to perform the authentication, but not for symmetric encryption using AES (i.e., hybrid solution).

B. Impact of security mechanism on the network layer

In this section, the impact of SAE on the system performance is analyzed by evaluating the Round Trip Time (RTT) and the code size of the IEEE 802.11s kernel module.

The first evaluation step is about evaluating the RTT – the time needed to send a request packet from node A to node B plus the time needed to transfer the response packet from node B back to node A. For this experiment five IEEE 802.11s nodes were used. Static paths through the network were defined to force multi-hop routes with different lengths (from direct connections to multi-hop routes with up to four hops). For each measurement with different path lengths involved and either SAE on or off, 10000 messages were sent from node A to node B. To measure the RTT, up to five BBBs (BBB0 to BBB4) were used to send UDP packets from BBB0 to BBB4 and back, using zero to three boards (BBB 1, 2, and 3) in between to forward the data. On BBB0 and BBB4, a UDP server-client application was used to i) send the request (BBB0), ii) receive this request and send the response (BBB4), and iii) to receive this response (BBB0). An adapted version of IEEE 802.11s was used to detect the exchanged UDP test packets. On BBB0 and BBB4 the received and sent timestamps were added to the test packets. These packets were collected on BBB0 during a test, and were used in a subsequent evaluation to compute the network layer RTT (by removing the time spent on application layer; only relevant for BBB0 and BBB4, as packet forwarding on BBBs 1-3 is only done on network layer).

In Figure 4 the results of these measurements are shown. The green, continuous line shows the RTT for different numbers of hops with SAE on and the blue, dashed line for the measured RTT with SAE disabled. Figure 4a shows that each hop significantly increases the RTT median (similar results were obtained when analyzing the average RTT). Additionally

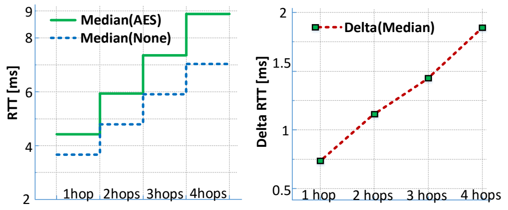


Fig. 4. Impact of SAE on network layer: SAE enabled vs. SAE disabled (i.e., None). Median (a), and delta(median)=median(SAE)-median(None) (b) of the RTT measurements using 10000 UDP packets are shown.

the delta of the median values is presented in Figure 4b: each hop increases the RTT, as a packet has to be encrypted and decrypted at each hop in between node A and node B.

In a second evaluation step, the impact of SAE on the code size of the MAC80211 kernel module was analyzed using the source code of this module (by default with SAE) of a standard 3.19 Linux kernel: first, the module with SAE included was compiled, and second, all the SAE-related source code was removed and the resulting module was compiled (and tested) once again. The results of the evaluation show that the impact of SAE on the code size of the IEEE 802.11s kernel module is about 5% w.r.t lines of code (LOC) (58545 LOC without SAE, 61354 LOC with SAE) and 3% w.r.t. the size in memory (774 KB without and 801 KB with SAE).

C. Impact of security mechanisms on the SW update duration

To secure and protect SecUp, different security mechanisms on network and on application layer are used. These mechanisms have an impact on the overall performance of the developed system. Latency measurements in the scope of real wireless SW updates using the Volvo FlexECU were performed to assess this impact. In the following, all steps required to successfully perform a wireless SW update using the developed framework are listed: i) WVI discovery and connection establishment, ii) authentication process between WVI and DT, iii) SW update initialization (mainly w.r.t. the ECU), iv) authorization on the ECU using a Seed&Key procedure, v) wireless data transfer from the DT to the WVI, vi) data download to the ECU via CAN, and vii) validation of the downloaded SW on the ECU.

Each step adds latency to the overall duration of a wireless SW update and the per-step-latency differs depending on the used security mechanisms. For this experiment the FlexECU and the corresponding binaries consisting of the application SW (378KB) plus the secondary bootloader (67KB) were used. On the application layer, security mechanisms implemented in SW utilizing the Java Bouncy Castle (JBC) were employed.

The gathered evaluation results presented in Table I reveal that i) the data transfer via CAN comprises most of the SW update duration (75% if all security mechanisms are enabled and up to 87% if disabled) and ii) that the employed security features increase the overall time required to carry out the wireless SW update by 20%. Although this duration increase is quite significant, it must be accepted as all security mechanisms are required to guarantee a secure system execution.

VIII. Framework Evaluation

In this section, the evaluation of SecUp and its wireless SW update mechanisms is presented. The performed evaluation is the first available analysis of a wireless SW update framework providing advanced SW update mechanisms as well as allowing secure and efficient SW updates in different (local) scenarios such as an assembly line or a service center. Therefore, no suitable benchmark is available to compare the gathered results. A basic system evaluation was already presented in [30]: a developed Vehicle and ECU Model (VEM) was used to perform fundamental experiments, system evaluations, and communication tests. Furthermore, the VEM was used to evaluate the behavior of the developed system in case of errors (e.g., sending unexpected frames) and communication problems (e.g., lost, delayed or duplicated CAN frames). This evaluation also included an analysis of the wireless network, where the DT is connected to a real vehicle and run wireless diagnostics using the developed WVI prototype. Thereby, the evaluation results were used to prove the applicability of IEEE 802.11s as media for wireless SW updates.

A. Wireless SW update analysis using Volvo FlexECU

The SW update process was first analyzed using the FlexECU, provided by Volvo Trucks and described in Section VI-C, connected to a WVI. This ECU is programmed in two steps: first, the secondary bootloader (SBL) is transferred to the ECU and then launched on the ECU using a specific UDS command. Second, the application binary is sent to the ECU and installed on it. SecUp supports both the use of an SBL and the application binary as well as an approach where the application binary is directly installed without using an SBL.

In Table II the duration of the wireless data transfer, where the SBL and the application SW is transferred from the DT to the WVI, is compared with time required to install the SBL and the application SW on the ECU via CAN. The measured SW update duration using the Volvo ECU is also used as a benchmark for the evaluation of AURIX and its implemented SW update features. The results reveal that the wireless data transfer (including data transfer, integrity check, etc.) is 12 times faster than forwarding the binary using CAN and thereby corresponds to the results presented in Table I. Thus, it makes sense that the WVI autonomously controls the data transfer to the ECU once the binary was received from the DT (i.e., no permanent wireless connection to the DT needed). The WVI then informs the DT when the SW is installed on the ECU successfully or when any problems occur.

B. Wireless SW update analysis using AURIX ECU

The AURIX ECU described in Section VI-D was used to further evaluate SecUp and its features. Therefore, the SW implementation of SecUp's core nodes, the DT and the WVI, were running on BBBs. These BBBs are connected to a measurement PC via USB to control the measurements and to collect the results. Per measurement campaign (i.e., one per mode) 20 wireless SW updates were performed and the duration of each single step, as already described in Section VII-C, was measured. These steps are then grouped in i) *Connection-related*, including the DT discovery and the authentication between DT and WVI, ii) *ECU-related*, encompassing the initialization of the SW update and the corresponding authorization step between WVI and ECU, iii) *Upload*, i.e., the wireless data transfer, and iv) *Download*, the data transfer via CAN as well as the installation of the SW on the ECU. This setup allows to compare the overall duration of a SW update w.r.t. the used update mechanism and additionally to analyze the latency added by each step.

To compare the SW update duration of both the Volvo FlexECU and the AURIX ECU, a SW binary for AURIX was created using Hightec Studio, the recommended SW development tool for AURIX. The resulting .hex file (i.e., the binary) was developed to have the same size as the Volvo FlexECU secondary bootloader (SBL) plus the size of the application SW (i.e., 67 KB + 378 KB = 445 KB). Although the .hex file contains all the information needed for the wireless SW update, the file format is not suitable to directly analyze the binary w.r.t. to partial SW updates and therefore a parser was developed to transform the .hex file in a so-called .phex (i.e., parsed hex) file format. In a .phex file, each line represents a block of the SW binary and it can therefore be used to check if a partial SW update is possible by comparing the lines of the SW versions. A further advantage of this file format is that the .phex file is smaller than the corresponding .hex file: the binary used for the evaluations is originally stored in a .hex file with 445 KB and can be transformed to a .phex file of about 316 KB.

1) Volvo FlexECU and AURIX ECU comparison: in a first evaluation step the SW update duration using both the FlexECU and AURIX are compared. The results presented in Table III show that AURIX can be updated more than twice as fast. This is due to: i) the initialization process on ECU level (i.e., start a programming session and handle authorization) is ten times faster on AURIX compared to the FlexECU due to the higher CPU power available, ii) the wireless data transfer is about 30% faster as the .phex is 29% smaller than the sum of SBL plus application SW, and iii) the data transfer on CAN (both ECUs are using a CAN bus with 500 Kbits/s) and the storage of the binary is close to three times faster on AURIX as the Volvo ECU first has to store and start the SBL and then load, store, and start the application SW.

2) Partial SW update evaluation: this experiment shows the benefits of a partial SW update compared to a traditional SW update. A SW update is often required due to a necessary bug fix or to update/adapt a parameter field of the ECU. In these cases, most parts of the SW remain unchanged and only some bytes have to be changed. To illustrate this, two different SW applications for AURIX were developed. The size of the .hex files of both versions is still 445 KB and they are utilizing a parameter field of 1024 byte in total. This parameter field is stored in a dedicated memory section of the AURIX ECU (see also Figure 3) and is the only difference between the SW versions. When utilizing partial SW updates, only the section containing the parameter field (1 KB) must be transferred to the ECU instead of transferring the entire binary.

In Table III the impact on the SW update duration is shown:

TABLE I
DURATION OF ALL REQUIRED STEPS IN MS OF A WIRELESS SW UPDATE W.R.T. SECURITY MECHANISMS ON NETWORK AND APPLICATION LAYER

Nw/Appl	Total	Discovery	Authentication	Init SW update	Seed&Key	Upload	Download	Validation
on/on	49193,3	3.2 (<0.1%)	2366,4 (4.8%)	1900,8 (3.9%)	509,8 (1.0%)	6585,4 (13.4%)	37315,3 (75.9%)	502,4 (1.0%)
off/on	47167,1	3.1 (<0.1%)	2360,4 (5.0%)	1992,0 (4.2%)	504,2 (1.1%)	5414,2 (11.5%)	36380,2 (77.1%)	505,1 (1.1%)
on/off	43745,6	3,2 (<0.1%)	1,88 (<0.1%)	1761,9 (4.0%)	510,7 (1.2%)	3756,0 (8.6%)	37200,2 (85.0%)	503,1 (1.2%)
off/off	41528,6	3.6 (<0.1%)	0,66 (<0.1%)	1764,67 (4.3%)	509,0 (1.2%)	2445,0 (5.9%)	36294,8 (87.4%)	502,7 (1.2%)

TABLE II
SW UPDATE DURATION: SW IS TRANSFERRED FROM A DT TO A WVI UTILIZING IEEE 802.11S AND THEN FORWARDED TO A ECU VIA CAN

Binary type	Binary size	On 11s	On CAN	Update duration
SBL	67KB	0.503s	6.268s	6.771s
Application	375 KB	2.527s	30.664s	33.191s

a partial SW update is about six times faster compared to the normal update using an AURIX ECU (i.e., a duration decrease of 83%). This is mainly due to the fact that less data has to be transferred wirelessly from the DT to the WVI (a speed-up of 98%) as well as from the WVI to the ECU via CAN (about twenty times faster). The benefit of a partial SW update, however, strongly depends on the memory architecture of an ECU as well as on the difference of two SW versions.

3) Parallel SW update evaluation: in the last evaluation step parallel SW updates are analyzed. These updates can be very beneficial in situations (see scenarios described in Section IV) where the same SW version shall be installed on ECUs integrated in several vehicles. This is of particular importance for the assembly line as well as for service centers when big vehicle recalls (e.g., due to a SW bug) are necessary, because the update can be performed on all vehicles simultaneously.

In the performed experiment two WVIs prototypes, each connected to an AURIX ECU, were wirelessly connected to the same DT. Instead of performing the SW update sequentially, the SW binary was installed on both ECUs in parallel.

The evaluation results presented in Table III show that the parallel update duration is increased by about 25% compared to a normal wireless SW update. This is due to the fact, that the required steps cannot by parallelized completely. However, parallel SW updates are still way faster (in this case 75% faster) compared to performing normal SW updates sequentially (i.e., performing a SW update two times in a row).

IX. CONCLUSION

In this paper SecUp, a generic framework enabling secure and efficient wireless SW updates is proposed. SecUp is designed and implemented to fulfill the requirements of several application scenarios: vehicle development, vehicle assembly line, vehicle maintenance, and OTA updates. SecUp encompasses efficient wireless SW update features such as parallel SW updates, where the SW of several vehicles is updated simultaneously, and partial SW updates, where only the changed parts of a SW binary are transferred and installed on an ECU. This paper also includes a description of the properties of the utilized IEEE 802.11s network and illustrates the designed cross-layer security concept used by SecUp.

A comparison of the developed SW update mechanisms show the benefits of utilizing advanced update features.

ACKNOWLEDGMENT

This work was partially funded by the SCOTT project, which received funding from the ECSEL Joint Undertaking under grant agreement No 737422. This joint undertaking is supported by EUs Horizon 2020 research and innovation program and Austria. SCOTT is also funded by the Austrian Federal Ministry BMVIT under the program "ICT of the Future" (05-2017 - 04-2020). The authors also acknowledge financial support by the COMET K2 Program of the BMVIT, BMWFW, FFG, the Province of Styria, and SFG.

REFERENCES

[1] Rudolf von Stokar, "Updating Car ECUs Over-The-Air (FOTA)," Red Bend Software, Tech. Rep. RedBend-01-2011, 2011.
[2] M. S. Idrees, H. Schweppe, Y. Roudier, M. Wolf, D. Scheuermann, and O. Henniger, "Secure automotive on-board protocols: A case of over-the-air firmware updates," Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), vol. 6596 LNCS, pp. 224–238, 2011.
[3] D. K. Nilsson and U. E. Larson, "Secure firmware updates over the air in intelligent vehicles," IEEE International Conference on Communications, pp. 380–384, 2008.
[4] D. Nilsson, P. Phung, and U. E. Larson, "Vehicle ECU classification based on safety-security characteristics," Road Transport Information and Control - RTIC, pp. 1–7, 2008.
[5] K. H. Peter Subke and V. Marquart, "ODX-based flash solution," CAN Newsletter, vol. 3, pp. 42–46, 2015.
[6] M. Khurram, H. Kumar, A. Chandak, V. Sarwade, N. Arora, and T. Quach, "Enhancing connected car adoption: Security and over the air update framework," pp. 194–198, Dec 2016.
[7] S. M. Mahmud, S. Shanker, and I. Hossain, "Secure software upload in an intelligent vehicle via wireless communication links," in Intelligent Vehicles Symposium. Proceedings. IEEE, June 2005, pp. 588–593.
[8] I. Hossain and S. M. Mahmud, "Analysis of a secure software upload technique in advanced vehicles using wireless links," in Intelligent Transportation Systems Conference, 2007. ITSC 2007. IEEE. IEEE, 2007, pp. 1010–1015.
[9] R. Petri, M. Springer, D. Zelle, I. McDonald, A. Fuchs, and C. Krauss, "Evaluation of lightweight tpms for automotive sw updates over the air," in Embedded Security in Cars Conference. escar, 2016.
[10] N. Gabe, "Over-the-air updates on varied paths," Automotive News, 2016-01-25.
[11] D. T. Tuan, S. Sakata, and N. Komuro, "Priority and admission control for assuring quality of I2V emergency services in VANETs integrated with Wireless LAN Mesh Networks," ICCE 2012, pp. 91–96, 2012.
[12] S. Chakraborty and S. Nandi, "IEEE 802.11s mesh backbone for vehicular communication: Fairness and throughput," IEEE Transactions on Vehicular Technology, vol. 62, no. 5, pp. 2193–2203, 2013.
[13] M. Steger, M. Karner, J. Hillebrand, W. Rom, E. Armengaud, M. Hansson, C. A. Boano, and K. Roemer, "Applicability of IEEE 802.11s for automotive wireless software updates," in 13th International Conference on Telecommunications (ConTEL), July 2015, pp. 1–8.
[14] W. K. Tan, S.-G. Lee, J. H. Lam, and S.-M. Yoo, "A security analysis of the 802.11s wireless mesh network routing protocol and its secure routing protocols," Sensors, vol. 13, no. 9, p. 11553, 2013.
[15] D. Harkins, "Simultaneous authentication of equals: A secure, password-based key exchange for mesh networks," in Second International Conference on Sensor Technologies and Applications. SENSORCOMM '08., Aug 2008, pp. 839–844.

TABLE III
DURATION OF ALL REQUIRED STEPS IN MS OF A WIRELESS SW UPDATE. COMPARISON BETWEEN THE VOLVO ECU AND THE AURIX MODES.

ECU/mode	Total	Connection-related	ECU-related	Upload	Download
Volvo/normal	48681.0	2369.7 (4.9%)	2410.6 (5.0%)	6585.4 (13.5%)	37315.3 (76.7%)
AURIX/normal	20768.8	2340.2 (11.3%)	205.4 (1.0%)	4597.0 (22.1%)	13626.3 (65.6%)
AURIX/partial	3570.7	2351.7 (65.9%)	216.7 (6.1%)	347.1 (9.7%)	655.2 (18.3%)
AURIX/parallel	25881.8	3868.3 (14.9%)	262.9 (1.0%)	7378.5 (28.5%)	14372.2 (55.5%)

[16] M. Sbeiti, A. Wolff, C. Wietfeld, and I. Technology, "PASER: Position Aware Secure and Efficient Route Discovery Protocol for Wireless Mesh Networks," in International Conference on Emerging Security Information, Systems and Technologies - SECURWARE, 2011, pp. 63–70.

[17] M. Sbeiti and C. Wietfeld, "One stone two birds: On the security and routing in wireless mesh networks," in Wireless Communications and Networking Conference (WCNC), 2014 IEEE, April 2014.

[18] M. S. Islam, M. A. Hamid, and C. S. Hong, "SHWMP: A Secure Hybrid Wireless Mesh Protocol for IEEE 802.11s Wireless Mesh Networks," in Transactions on Computational Science VI. Springer Berlin Heidelberg, 2009, pp. 95–114.

[19] J. Ben-Othman and Y. I. Saavedra Benitez, "IBC-HWMP: a novel secure identity-based cryptography-based scheme for Hybrid Wireless Mesh Protocol for IEEE 802.11s," Concurrency and Computation: Practice and Experience, vol. 25, no. 5, pp. 686–700, 2013.

[20] M. Steger, M. Karner, J. Hillebrand, W. Rom, and K. Roemer, "A Security Metric for Structured Security Analysis of Cyber-Physical Systems Supporting SAE J3061," pp. 1–8, 2016.

[21] SAE, "SAE J3061: Surface Vehicle Recommended Practice - Cybersecurity Guidebook for Cyber-Physical Vehicle Systems," SAE International, Tech. Rep. 01-2016, 2016.

[22] "open80211s – An open-source implementation of the recently ratified IEEE 802.11s wireless mesh standard," https://github.com/o11s/open80211s/wiki/, accessed: 2017-11-30.

[23] ISO, "ISO 22901-1: Road vehicles – Open diagnostic data exchange (ODX) – Part 1: Data model specification," ISO, Tech. Rep. 2008-11, 2008.

[24] ——, "ISO 14229: Road vehicles – Unified diagnostic services (UDS) – Specification and requirements," ISO, Tech. Rep. 14229:2006(E), 2006.

[25] I. Garitano, S. Fayyad, and J. Noll, "Multi-Metrics Approach for Security, Privacy and Dependability in Embedded Systems," Wirel. Pers. Commun., vol. 81, no. 4, pp. 1359–1376, Apr. 2015.

[26] B. Potter, "Microsoft SDL Threat Modelling Tool," Network Security, vol. 2009, no. 1, pp. 15 – 18, 2009.

[27] M. Steger, C. Boano, M. Karner, J. Hillebrand, W. Rom, and K. Roemer, "SecUp: Secure and Efficient Wireless Software Updates for Vehicles," pp. 628–636, 2016.

[28] M. Steger, A. Dorri, S. S. Kanhere, K. Roemer, R. Jurdak, and M. Karner, Secure Wireless Automotive Software Updates Using Blockchains: A Proof of Concept. Cham: Springer International Publishing, 2018, vol. 23, pp. 137–149.

[29] ISO, "ISO 26262: Road vehicles – Functional safety – Part 1: Vocabulary," ISO, Tech. Rep. 26262-1:2011, 2011.

[30] M. Steger, M. Karner, J. Hillebrand, W. Rom, C. Boano, and K. Roemer, "Generic framework enabling secure and efficient automotive wireless SW updates," pp. 1–8, Sept 2016.

Marco Steger received the Masters degree in Information and Computer Engineering from Graz University of Technology, Graz, Austria, in 2013, with a thesis titled "Development and Evaluation of C2X Applications" done in cooperation with BMW, Munich, Germany.

He is currently a Senior Researcher with the VIRTUAL VEHICLE Research Center in Graz, Austria. His research interests include wireless automotive communication networks, security in wireless/mobile/automotive networks, wireless sensor networks, and Car-to-X applications.

Carlo Alberto Boano (M2009) received the double Masters degree in Computer Engineering from Politecnico di Torino, Turin, Italy, and KTH Stockholm, Stockholm, Sweden in 2009, and the Doctoral degree sub-auspiciis praesidentis in Information and Communication Technology from Graz University of Technology, Graz, Austria, in 2016, with a thesis on dependable WSNs.

He is currently an Assistant Professor with the Institute for Technical Informatics of Graz University of Technology, Graz, Austria. His research interests encompass the design of dependable networked embedded systems and the robustness of IoT communication protocols against environmental influences.

Thomas Niedermayr received the Masters degree in Information and Computer Engineering from Graz University of Technology, Graz, Austria, in 2017, with a thesis titled "Enabling Wireless Automotive SW Updates for the Infineon AURIX ECU".

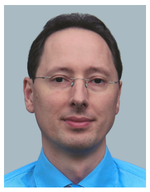
His current research focuses on the development of Embedded and Automotive systems.

Kay Roemer received the Dipl.-Inf. degree in Computer Science from University of Frankfurt/Main, Germany, in 1999, and the Doctoral (Dr.sc.ETH) degree in Computer Science from ETH Zurich, Zurich, Switzerland, in 2005.

Kay Roemer is currently a Professor and the Director with the Institute for Technical Informatics, Graz University of Technology, Graz, Austria. He was a Professor with the University of Luebeck, Luebeck, Germany, and a Senior Researcher with ETH Zurich, Zurich, Switzerland. His research interests include wireless networking, fundamental services, operating systems, programming models, dependability, and deployment methodology of networked embedded systems, in particular IoT, cyber-physical systems, and sensor networks.

Karner Michael received the Ph.D. degree in Electrical Engineering from Graz University of Technology, Graz, Austria in 2011.

Since 2011, he has been with VIRTUAL VEHICLE, Graz, Austria. As Lead Researcher and Project Manager, he is working in several large international industrial research projects. His research interests include wireless technologies, cyber-physical systems, functional and active safety - especially with focus on simulation/co-simulation based analysis and verification.



Joachim Hillebrand studied Electrical and Communications Engineering at Graz University of Technology, Austria, where he received his Dipl.-Ing. degree in 2000.

Today, he leads the Embedded Systems Group at VIRTUAL VEHICLE, Graz, Austria. Before joining VIRTUAL VEHICLE in 2009, he held research positions with NTT DoCoMo Eurolabs and BMW Research and Technology, Munich, Germany. His research interests include automotive E/E architectures as well as wireless and wired vehicle data communication.

Bibliography

[1] I. 14229:2006(E). ISO 14229:2006: Road vehicles – Unified diagnostic services (UDS) – Specification and requirements, 2006.

[2] I. T. G. 802.11n. Local and metropolitan area networks. Amendment to Part 11: Amendment 11n: Enhancements for Higher Throughput Mesh Networking. Technical report, IEEE, 2009.

[3] C. E. Andrade, S. D. Byers, V. Gopalakrishnan, E. Halepovic, M. Majmundar, D. J. Poole, L. K. Tran, and C. T. Volinsky. Managing massive firmware-over-the-air updates for connected cars in cellular networks. In *Proceedings of the 2Nd ACM International Workshop on Smart, Autonomous, and Connected Vehicular Systems and Services*, CarSys '17, pages 65–72, New York, NY, USA, 2017. ACM.

[4] A. Avizienis, J.-C. Laprie, B. Randell, and C. Landwehr. Basic concepts and taxonomy of dependable and secure computing. *Dependable and Secure Computing, IEEE Transactions on*, 1(1):11–33, Jan 2004.

[5] P. Azzoni, F. Rogo, C. Coveri, M. Steger, W. Rom, A. Fiaschetti, F. Liberati, and J. Noll. Applying shield in new domains. In A. Fiaschetti, J. Noll, P. Azzoni, and R. Uribeetxeberria, editors, *Measurable and Composable Security, Privacy, and Dependability for Cyberphysical Systems: The SHIELD Methodology*, pages 415–452. CRC Press, 2017.

[6] B. Steurich and M. Klimke and Ines Pedersen. Automotive ecus: Architecture considerations to implement secure software updates over the air. In *White Paper Infineon Technologies published in Embedded computing (Online Resource; last visited 2018-01-22)*, 2017.

[7] S. Brown and C. J. Sreenan. Software updating in wireless sensor networks: A survey and lacunae. *Journal of Sensor and Actuator Networks*, 2(4):717–760, 2013.

[8] M. Cesana, L. Fratta, M. Gerla, E. Giordano, and G. Pau. C-vet the ucla campus vehicular testbed: Integration of vanet and mesh networks. In *Proc. of the European Wireless Conference*, 2010.

[9] R. N. Charette. This car runs on code. *IEEE spectrum*, 46(3):3, 2009.

[10] M. L. Chiang and T. L. Lu. Two-stage diff: An efficient dynamic software update mechanism for wireless sensor networks. In *2011 IFIP 9th International Conference on Embedded and Ubiquitous Computing*, pages 294–299, 2011.

[11] Cozybit. open80211s – An open-source implementation of the recently ratified IEEE 802.11s wireless mesh standard. `http://open80211s.org/open80211s/`, 2015.

[12] A. Dorri, S. S. Kanhere, and R. Jurdak. Towards an optimized blockchain for iot. In *Proceedings of the Second International Conference on Internet-of-Things Design and Implementation*, pages 173–178. ACM, 2017.

[13] A. Dorri, M. Steger, S. S. Kanhere, and R. Jurdak. Blockchain: A distributed solution to automotive security and privacy. *IEEE Communications Magazine*, 55(12):119–125, December 2017.

[14] U. Drolia, Z. Wang, S. Vemuri, M. Behl, and R. Mangharam. Autoplug – an automotive test-bed for electronic controller unit testing and verification. In *Proc. of the IEEE Intelligent Transportation Systems Conference (ITSC)*, 2011.

[15] Drozhzhin, Alex. Tesla Model S was hacked remotely. *Kaspersky lab (online blog; last accessed 2018-01-13*, 2016.

[16] M. J. Dworkin. Recommendation for block cipher modes of operation: Galois/counter mode (gcm) and gmac. *Special Publication (NIST SP)-800-38D*, 2007.

[17] X. Fan, F. Susan, W. Long, and S. Li. Security analysis of zigbee. *White Paper*, 2017.

[18] J. Fitzpatrick. The Difference Between WEP, WPA, and WPA2 Wi-Fi Passwords. *How-To Geek (Online resource; last visited 2017-12-10)*, September 21, 2016.

[19] N. Gabe. Over-the-Air Updates on Varied Paths. *Automotive News*, 2016.

[20] J. Gapper. Software is steering auto industry. *Financial Times*, February 18, 2015.

[21] I. Garitano, S. Fayyad, and J. Noll. Multi-metrics approach for security, privacy and dependability in embedded systems. *Wirel. Pers. Commun.*, 81(4):1359–1376, Apr. 2015.

[22] Greenberg, Andy. Hackers Cut a Corvette's Brakes Via a Common Car Gadget. *Wired (online magazine; last accessed 2017-12-13*, 2015.

[23] S. Gupta and R. Dham. Improving Security with Bluetooth Low Energy 4.2, 2016.

[24] P. Hank, S. Müller, O. Vermesan, and J. Van Den Keybus. Automotive ethernet: In-vehicle networking and smart mobility. In *Proceedings of the Conference on Design, Automation and Test in Europe*, DATE '13, pages 1735–1739, San Jose, CA, USA, 2013. EDA Consortium.

[25] D. Harkins. Simultaneous Authentication of Equals: A Secure, Password-Based Key Exchange for Mesh Networks. In *Second International Conference on Sensor Technologies and Applications. SENSORCOMM '08.*, pages 839–844, Aug 2008.

[26] I. Hossain and S. Mahmud. Analysis of a secure software upload technique in advanced vehicles using wireless links. In *Proc. of the Intelligent Transportation Systems Conference*, 2007.

[27] I. Hossain, S. Mahmud, and M. Hwang. Performance evaluation of mobile multicast session initialization techniques for remote sw upload in vehicle ecus. In *Proc. of the IEEE Vehicular Technology Conference*, 2010.

[28] I. Hossain and S. M. Mahmud. Analysis of group key management protocols for secure multicasting in vehicular software distribution network. In *Third IEEE International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob 2007)*, pages 25–25, Oct 2007.

[29] IEEE. Local and metropolitan area networks-Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY):Amendment 10: Mesh Networking. Technical report, IEEE, 2011.

[30] IEEE. Standard for local and metropolitan area networks–part 15.4: Low-rate wireless personal area networks (lr-wpans). Technical report, IEEE, Sept 2011.

[31] Infineon. Automotive Security – Trusted Driving. `https://www.infineon.com/cms/en/applications/automotive/automotive-security/`. Accessed: 2018-03-05.

[32] ISO. ISO 22901-1: Road vehicles – Open diagnostic data exchange (ODX) – Part 1: Data model specification. Technical report, ISO, 2008.

[33] ISO. ISO 26262: Road vehicles – Functional safety – Part 1: Vocabulary. Technical report, ISO, 2011.

[34] Kemp, Leanne . Putting bling on the blockchain: The Everledger stor. *Swiss Re Institute (online resource; last accessed 2018-01-13*, 2017.

[35] H. Liu, X. Liang, L. Fang, B. Zhang, and J. wen Zhao. A secure and efficient authentication protocol based on identity based aggregate signature for electric vehicle. In *Wireless Communication and Sensor Network (WCSN), 2014 International Conference on*, pages 353–357, Dec 2014.

[36] M. Khurram et al. Enhancing connected car adoption: Security and ota update framework. In *Proc. of the 3rd World Forum on Internet of Things*, 2016.

[37] M. Steger et al. An Efficient and Secure Automotive Wireless Software Update Framework. *Under submission*, 2017.

[38] A. Mahmood, N. Javaid, and S. Razzaq. A review of wireless communications for smart grid. *Renewable and Sustainable Energy Reviews*, 41(Supplement C):248 – 260, 2015.

[39] S. M. Mahmud, S. Shanker, and I. Hossain. Secure software upload in an intelligent vehicle via wireless communication links. In *Intelligent Vehicles Symposium. Proceedings. IEEE*, pages 588–593, June 2005.

[40] P. Morgner, S. Mattejat, Z. Benenson, C. Mueller, and F. Armknecht. Insecure to the Touch: Attacking ZigBee 3.0 via Touchlink Commissioning. In *Proceedings of the 10th ACM Conference on Security and Privacy in Wireless and Mobile Networks*, WiSec '17, pages 230–240, New York, NY, USA, 2017. ACM.

[41] M.S. Idrees et al. Secure Automotive On-board Protocols: A Case of Over-the-air Firmware Updates. In *Proc. of the 3rd Conference on Communication Technologies for Vehicles (Nets4Cars/Nets4Trains)*, 2011.

[42] S. Nakamoto. Bitcoin: A peer-to-peer electronic cash system, 2008.

[43] D. Nilsson and U. Larson. Secure firmware updates over the air in intelligent vehicles. *IEEE Conference on Communications*, 2008.

[44] D. Nilsson, P. Phung, and U. E. Larson. Vehicle ECU classification based on safety-security characteristics. *Road Transport Information and Control - RTIC*, pages 1–7, 2008.

[45] J. Noll, I. Garitano, S. Fayyad, E. Asberg, and H. Abie&. Measurable security, privacy and dependability in smart grids. *Journal of Cyber Security*, 3:371–398, 2015.

[46] Y. Onuma, M. Nozawa, Y. Terashima, and R. Kiyohara. Improved software updating for automotive ecus: Code compression. In *2016 IEEE 40th Annual Computer Software and Applications Conference (COMPSAC)*, volume 2, pages 319–324, June 2016.

[47] K. H. Peter Subke and V. Marquart. ODX-based flash solution. *CAN Newsletter*, pages 42–46, 2015.

[48] B. Potter. Microsoft SDL threat modelling tool. *Network Security*, pages 15–18, 2009.

[49] R. Petri et al. Evaluation of lightweight tpms for automotive sw updates over the air. In *Proc. of the Conference on Embedded Security in Cars*, 2016.

[50] Redbend Software. Updating Car ECUs Over-The-Air (FOTA). *White Paper*, page 14, 2011.

[51] Rudolf von Stokar. Updating Car ECUs Over-The-Air (FOTA), 2013.

[52] SAE. SAE J3061: Surface Vehicle Recommended Practive - Cybersecurity Guidebook for Cyber-Physical Vehicle Systems. Technical report, SAE International, 2016.

[53] A. Sari and M. Karay. Comparative Analysis of Wireless Security Protocols: WEP vs WPA. In *International Journal of Communications, Network and System Sciences*, volume 9, 2015.

[54] J. Scarpati. Wireless security protocols: The difference between WEP, WPA, WPA2. *TechTarget (Online resource; last visited 2017-12-10)*, January, 2017.

[55] G. Shi, Z. Ke, F. Yan, J. Hu, W. Yin, and Y. Jin. A vehicle electric control unit over-the-air reprogramming system. In *International Conference on Connected Vehicles and Expo (ICCVE)*, pages 48–51, Oct 2015.

[56] B. SIG. Specification of the Bluetooth System. Version 4.2. Architecture & Terminology Overview. Technical report, Bluetooth SIG, 2014.

[57] M. Steger, C. Boano, M. Karner, J. Hillebrand, W. Rom, and K. Roemer. SecUp: Secure and Efficient Wireless Software Updates for Vehicles. In *Proc. of the Conference on Digital System Design (DSD)*, Limassol, Cyprus, 2016. IEEE.

[58] M. Steger, C. A. Boano, K. Römer, M. Karner, J. Hillebrand, and W. Rom. CESAR: a Testbed Infrastructure to Evaluate the Efficiency of Wireless Automotive Software Updates. In *Proceedings of the 20^{th} ACM International Conference on Modeling, Analysis and Simulation of Wireless and Mobile Systems (MSWiM)*, pages 311–315. ACM, Nov. 2017.

[59] M. Steger, A. Dorri, S. S. Kanhere, K. Roemer, R. Jurdak, and M. Karner. Secure wireless automotive software updates using blockchains: A proof of concept. In C. Zachaeus, B. Mueller, and G. Meyer, editors, *Advanced Microsystems for Automotive Applications 2017: Smart Systems Transforming the Automobile*, pages 137–149. Springer International Publishing, Cham, 2018.

[60] M. Steger, M. Karner, J. Hillebrand, W. Rom, E. Armengaud, M. Hansson, C. Boano, and K. Roemer. Applicability of IEEE 802.11s for Automotive Wireless SW Updates. In *Proc. of the 13th Conference on Telecommunications*, 2015.

[61] M. Steger, M. Karner, J. Hillebrand, W. Rom, C. Boano, and K. Roemer. Generic Framework Enabling Secure and Efficient Automotive Wireless SW Updates. In *Proc. of the Conf. on Emerging Technologies and Factory Automation*, 2016.

[62] M. Steger, M. Karner, J. Hillebrand, W. Rom, and K. Roemer. A Security Metric for Structured Security Analysis of Cyber-Physical Systems Supporting SAE J3061. In *CPSData – Second International Workshop on modeling, analysis and control of complex Cyber-Physical Systems*, pages 1–8, 2016.

[63] Valasek, Chris and Miller, Charlie . Remote Exploitation of an Unaltered Passenger Vehicle. *White Paper*, page 93, 2015.

[64] W. Vandenberghe, I. Moerman, P. Demeester, and H. Cappelle. Suitability of the wireless testbed w-ilab.t for vanet research. In *Proc. of the 18th Symposium on Communications and Vehicular Technology in the Benelux*, 2011.

[65] N. Vidgren, K. Haataja, J. L. Patino-Andres, J. J. Ramírez-Sanchis, and P. Toivanen. Security Threats in ZigBee-Enabled Systems: Vulnerability Evaluation, Practical Experiments, Countermeasures, and Lessons Learned. In *46th Hawaii International Conference on System Sciences*, pages 5132–5138, Jan 2013.

[66] R. W. World. Bluetooth vs BLE – difference between Bluetooth and BLE (Bluetooth Low Energy). `http://www.rfwireless-world.com/Terminology/Bluetooth-vs-BLE.html`. Accessed: 2017-11-13.