THESIS FOR THE DEGREE OF DOCTOR OF PHILOSOPHY

Towards Secure and Forensically-Enabled Resilient Vehicle Design

Kim Strandberg

Division of Computer and Network Systems
Department of Computer Science & Engineering
Chalmers University of Technology
Gothenburg, Sweden, 2025

Towards Secure and Forensically-Enabled Resilient Vehicle Design

Kim Strandberg

ISBN: 978-91-8103-204-8
Doktorsavhandlingar vid Chalmers tekniska högskola
Ny serie nr 5662
ISSN 0346-718X

Department of Computer Science & Engineering
Division of Computer and Network Systems
Chalmers University of Technology
Gothenburg, Sweden

Author e-mail: kim.strandberg@volvocars.com

This thesis has been prepared using LaTeX.
Printed by Chalmers Digitaltryck,
Gothenburg, Sweden, 2025.

"The best way to predict the future is to invent it."
- Alan Kay

Towards Secure and Forensically-Enabled Resilient Vehicle Design

Kim Strandberg
Department of Computer Science and Engineering,
Chalmers University of Technology

Abstract

The rise of autonomous and connected vehicles has introduced significant cybersecurity challenges in the automotive domain. An increase in regulations has mandated compliance with vehicle cybersecurity requirements. These regulations require vehicles to be designed to withstand cyberattacks, equipped with mechanisms to detect and effectively respond to threats, and ensure a secure process for software updates and digital forensics. However, a gap remains in providing clear technical guidance for securing vehicles and ensuring compliance with evolving regulations. This thesis aims to address this gap by presenting tools and methodologies to strengthen cybersecurity within the automotive industry.

In the first part of the thesis, we analyze and adapt methodologies for various phases of the vehicle life cycle and propose a systematic approach to predict and mitigate vulnerabilities throughout the entire life cycle. We also conduct a comprehensive review of resilience techniques, fault tolerance, and dependability related to attack detection, mitigation, recovery, and endurance. By applying our methodology and integrating these review findings, we develop a framework to design vehicles that are safe, secure, and resilient against various cyberattacks. In addition, we perform a systematic literature review of automotive digital forensics, providing an overview of the research landscape and its practical applications. This review guides future research and supports engineers in developing forensic mechanisms.

The second part focuses on architecture, where we introduce a reference architecture for vehicle software updates to address the growing need for rapid and secure bug patching and software modifications. We present an attacker model, perform a threat assessment, define general security requirements that align with common security goals and directives, and provide formal proof of security and correctness. Furthermore, we propose a second reference architecture that addresses the digital forensic challenges identified in the first part of the thesis, with the aim of improving the security and effectiveness of forensic practices within the automotive domain.

In summary, this thesis presents tools and methodologies to strengthen cybersecurity in the automotive domain and guide compliance with regulations. It provides a proactive approach to predict and mitigate vehicle vulnerabilities, integrates resilience techniques into vehicle design, establishes a secure software update framework, and offers insights and guidelines for designing automotive digital forensic systems.

Keywords: automotive, security, resilience, forensics, software updates

Acknowledgment

First and foremost, I would like to express my deepest gratitude to my family, especially my wife, Lisa, and my four children, Jakob, Elias, Isak, and Simon, for your love, understanding, and support. Thank you for being my constant source of strength and inspiration throughout this journey. I also would like to express my heartfelt gratitude to our dog, Tove. Your presence and warmth, especially during late-night sessions in front of the computer, have provided comfort and companionship.

I want to thank my supervisors, Tomas Olovsson, Ulf Arnljung, Nasser Nowdehi, Magnus Almgren, and my examiner, Per Larsson-Edefors, for continuous and constructive feedback and all co-authors for fruitful discussions and feedback. I would also like to thank all industrial partners involved in the CyReV project, my formers and current group and product managers at Volvo Cars, Ulf Edvardsson, Karolina Hill, Cristina Cristescu Dalmasso, Babar Farooq, and Hans Alminger, and all my colleagues at Volvo Cars and Chalmers for your support and encouragement during my studies.

Lastly, I would like to thank Volvo Cars and VINNOVA, the Swedish Governmental Agency for Innovation Systems, for funding my research within the CyReV project (2019-03071).

Kim Strandberg
Gothenburg, May 2025

List of Publications

Appended publications

This thesis is based on the following publications:

Guiding Automotive Cybersecurity, Resilience, and Digital Forensics: Frameworks and Principles

[A] **K. Strandberg**, T. Olovsson, E. Jonsson, "Securing the Connected Car: A Security-Enhancement Methodology" *IEEE Vehicular Technology Magazine, 2018.*

[B] T. Rosenstatter, **K. Strandberg**, R. Jolak, R. Scandariato, T. Olovsson "REMIND: A Framework for the Resilient Design of Automotive Systems" *IEEE Secure Development, 2020.*

[C] **K. Strandberg**, T. Rosenstatter, R. Jolak, N. Nowdehi, T. Olovsson "Resilient Shield: Reinforcing the Resilience of Vehicles Against Security Threats" *IEEE Vehicular Technology Conference, 2021.*

[D] **K. Strandberg**, N. Nowdehi, T. Olovsson "A Systematic Literature Review on Automotive Digital Forensics: Challenges, Technical Solutions and Data Collection" *IEEE Transactions on Intelligent Vehicles, 2023.*

Reference Architectures: Secure Automotive Software Updates and Digital Forensics

[E] **K. Strandberg**, D. K. Oka, T. Olovsson "UniSUF: A Unified Software Update Framework for Vehicles Utilizing Isolation Techniques" *19th escar Europe: The World's Leading Automotive Cyber Security Conference, 2021.*

[F] **K. Strandberg**, U. Arnljung, T. Olovsson, D. K. Oka "Secure Vehicle Software Updates: Requirements for a Reference Architecture" *IEEE Vehicular Technology Conference, 2023.*

[G] **K. Strandberg**, U. Arnljung, T. Olovsson "The Automotive Black-Box: Towards a Standardization of Automotive Digital Forensics" *IEEE International Workshop on Information Forensics and Security, 2023.*

Under submission

[H] M. S. Hagen, E. Lundqvist, A. Phu, Y. Wang, **K. Strandberg**, E. M. Schiller “Towards a Formal Verification of Secure Vehicle Software Updates”

Other publications

The following publication was either published or submitted during my Ph.D. studies but is not included in the thesis.

R. Jolak, T. Rosenstatter, M. Mohamad, **K. Strandberg**, B. Sangchoolie, N. Nowdehi, R. Scandariato “CONSERVE: A framework for the selection of techniques for monitoring containers security” *Journal of Systems and Software, Volume 186, 2022.*

Under submission

K. Strandberg and M. Eldefrawy “Advances in Automotive Digital Forensics: Recent Trends and Future Directions”

Contents

Chapter 1

Introduction

1.1 Evolution of the Automotive Industry

In recent years, vehicles have evolved into sophisticated Cyber-Physical Systems (CPS), transforming from simple transportation vessels to computerized entities with over 150 computers and more than 100 million lines of software code, and by 2030, it is expected to rise to 300 million lines of code [1]. These systems control critical functions, including steering, braking, and engine control.

Figure 1.1 shows the Volvo ÖV4 (Open Carriage), known as Jakob, which was named after the Swedish calendar name day on which the model was completed, July 25, 1926. This vehicle marks Volvo's first entry into car manufacturing [2]. A vehicle like this, naturally, offers strong cybersecurity due to its limited technology and connectivity. However, reducing functionality to improve cybersecurity in modern vehicles is rarely justified or acceptable.

Figure 1.1: The Volvo car model, commonly referred to as Jakob [3]

To better understand the principles of modern vehicle architecture, a simplified illustration is presented in Figure 1.2. In Chapter 5, we categorize vehicle electronics into four main groups: *internal and external communication*,

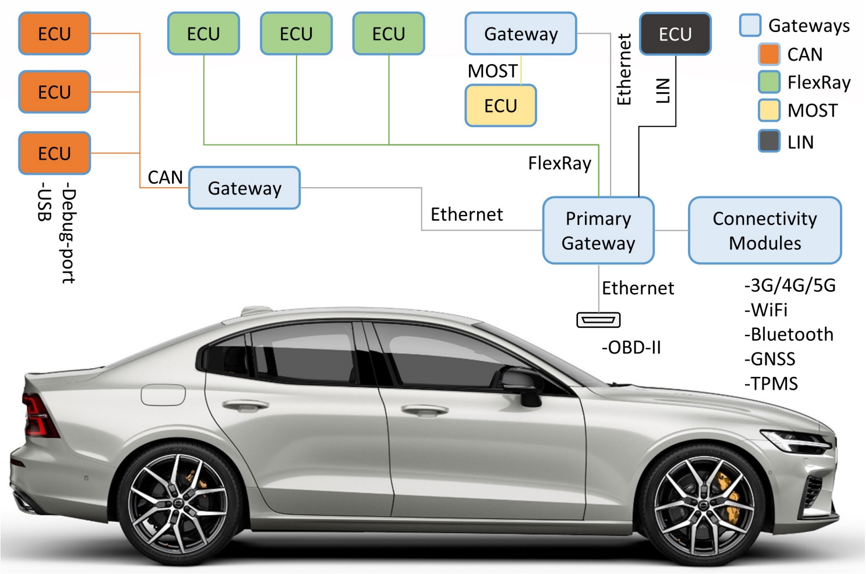


Figure 1.2: An example of vehicle architecture, with an image of a Volvo car [3, 4].

hardware, *software*, and *data storage*. As shown in Figure 1.2, the first group involves communication buses, including CAN, FlexRay, MOST, and LIN. These are utilized in various network segments according to their specific characteristics. For example, FlexRay is often used for safety-critical systems because of its speed and reliability compared to CAN, while MOST is designed for media-oriented data. Typically, a primary gateway and several additional gateways translate and transmit data between different segments. Additionally, vehicles have multiple connection points and communication interfaces to external devices, such as USB ports, WiFi, Bluetooth, and 4G/5G.

The second group, hardware, consists of *Electronic Control Modules (ECUs)*, *sensors*, and *actuators*. The complexity of an ECU varies according to its function, ranging from basic processing of sensor signals to managing an infotainment system with numerous applications. Traditional mechanical linkages, which physically connect components such as the steering to the wheels or the gas pedal to the throttle, are being replaced by *Drive-by-Wire (DbW)* systems. These systems use sensors, actuators, and ECUs to electronically manage safety critical functions without the need for direct mechanical connections. Sensors provide various data, including speed, temperature, distance, and detection of obstacles such as pedestrians and animals, and include laser and ultrasonic devices, as well as cameras. As visualized in Figure 1.3, the actuators use the input of the sensors, processed and directed by the ECU, to perform specific actions, such as braking, steering, and engine control.

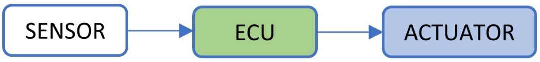


Figure 1.3: Input from sensors is received and processed by Electronic Control Units (ECUs), which then generate signals for specific actions through actuators.

The third group, software, includes software installed or actively running in ECUs, thus involving transit, at-rest, and running states. Additionally, the category includes software update systems such as over-the-air (OTA) and workshop updates.

Finally, the fourth group, data storage, comprises various types of data, including forensic logs, fault codes, software update reports, previously executed diagnostics, and cryptographic keys.

From an infrastructure perspective, the advancement of *Cooperative Intelligent Transport Systems (C-ITS)* has facilitated *vehicle-to-everything (V2X) communication*. This communication includes interactions not only between vehicles but also with the broader infrastructure, cloud services, and other road users like pedestrians and cyclists, creating a dynamic ecosystem where vehicles contribute to and benefit from a wealth of shared information. As shown in Figure 1.4, vehicles can serve as nodes within a more extensive system, within smart cities, or in advanced infrastructure setups. Through V2X communication, these vehicles exchange important data such as locations, traffic conditions, and information about approaching vehicles. This data exchange facilitates real-time updates on traffic flow and potential hazards, such as accidents or roadblocks, and contributes to safer and more efficient travel. In addition to physical objects, such as traffic lights, road signs, and roadside units (RSUs), virtual entities, like virtual traffic lights, can provide additional information. Integrating virtual entities into C-ITS improves flexibility, awareness, and intelligence in traffic management, thus improving road safety. For example, by providing real-time information on vehicle locations, weather conditions, accidents, and traffic congestion, travel routes and vehicle speeds can be dynamically adjusted to different situations. This results in improved

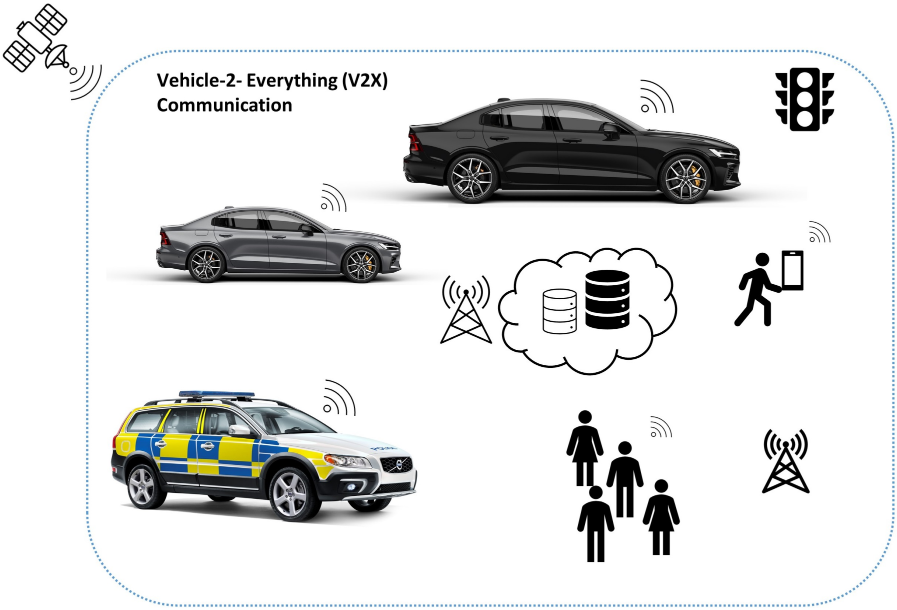


Figure 1.4: An example of V2X communication, with images of Volvo cars [3, 4].

traffic flow, decreased travel times, and increased efficiency in transportation systems.

Establishing trust in communication and guaranteeing the authenticity of the signals, processing, and resulting actions is vital. Additionally, ensuring secure storage and trustworthy data is imperative, as data manipulation can have severe consequences. For instance, granting attackers extended privileges to install a backdoor for remote access, tampering with or deleting digital evidence, or altering logs that could affect upcoming software updates, such as blocking vulnerability patches, can allow exploitable issues to persist.

Thesis Organization. This thesis is structured as follows: Section 1.2 details the challenges of securing vehicles. Section 1.3 provides the necessary background information on key topics, including automotive cybersecurity and resilience, secure software updates, and automotive digital forensics. Section 1.4 outlines the thesis objectives, contributions, and research methodologies, while Section 1.5 offers an overview of the publications. Section 1.6 presents proposals for future work, and Section 1.7 provides a summary and conclusions of the thesis. Finally, the appended publications are included.

1.2 Challenges and Motivation

We are transitioning into a hybrid world that integrates virtual and physical entities, and while these advances, as mentioned in Section 1.1, present opportunities, they also raise significant security concerns. While safety has long been prioritized within the automotive industry [5], automotive cybersecurity is relatively new [6].

The complexity of modern vehicles not only increases the risk of vulnerabilities but also expands the attack surface due to greater connectivity. This amplifies the potential to exploit these vulnerabilities, making vehicle security both complex and challenging. As shown in Chapter 4 through numerous incidents, modern vehicles are susceptible to cyberattacks. Thus, novel approaches are necessary to identify, analyze, and mitigate vulnerabilities. Although it may not be feasible to address all vulnerabilities, it is crucial to establish strategies to create a baseline for protection against common security threats and attacks. However, cyberattacks can occasionally succeed, highlighting the need to strengthen cybersecurity with a resilience approach to ensure safe operations even in the event of a successful breach.

There are various challenges within the automotive domain related to cybersecurity. We categorize some examples of these challenges into three key areas below.

Technical issues: inherent limitations of existing systems and protocols used in automotive technology.

-Lack of Encryption and Authentication. The bus technologies discussed in Section 1.1 focus mainly on safety and reliability rather than cybersecurity. For example, CAN nodes rely on broadcasting for communication without encryption or message authentication, making them susceptible to malicious devices capable of recording, manipulating, or spoofing messages while posing as trusted entities.

-Denial of Service (DoS) Attacks. CAN's priority-based system makes

it susceptible to DoS attacks, where malicious actors can disrupt vehicle communication by flooding the system with illegitimate, high-priority messages.

-*Performance Constraints of ECUs.* The limited performance capabilities of many modern vehicle ECUs pose challenges in implementing security measures such as cryptographic operations.

-*Bandwidth Constraints in Communication Protocols.* Bandwidth constraints in automotive communication protocols further complicate efforts to address security concerns, as increased security measures can reduce system efficiency and slow communication due to the larger data payloads required for encryption, authentication, and similar processes.

-*Real-Time Requirements.* A vehicle is a safety-critical system with real-time requirements, meaning that implemented security measures cannot introduce delays that conflict with these requirements.

-*Software Updates.* Ensuring secure software updates for automotive is challenging due to the complex vehicular system along with a dynamic threat landscape (cf. Section 1.3.2).

-*Liability and Accountability.* Ensuring trust and automating the collection of relevant information from large volumes of data to enable traceability in digital forensic investigations is challenging (cf. Section 1.3.3).

Design and Integration Issues: cost and technical compatibility in both existing and future automotive architectures.

-*Cost.* A key challenge is the cost of integrating more advanced security hardware into new vehicle designs. Manufacturers must balance the need for enhanced cybersecurity with economic feasibility.

-*Legacy System Compatibility.* Ensuring that new cybersecurity technologies are compatible with legacy systems is another significant issue. Integrating modern security measures into older systems without negatively affecting existing functionality is challenging.

-*Supply Chain.* The complexity of managing the automotive supply chain, with its many layers of software and hardware suppliers, presents additional challenges. Ensuring consistent security across all suppliers is challenging but essential for securing automotive systems.

Regulatory and Compliance issues: ensuring compliance with evolving regulations and addressing long-term security concerns.

-*Regulatory Compliance.* Regulatory compliance introduces its own complexities. For instance, aligning the data collection requirements for Event Data Recorders (EDRs) [7] and digital forensics [8] with GDPR regulations [9] while also meeting real-time safety and cybersecurity needs is a significant challenge for manufacturers.

Vehicle Lifespan. Ensuring cybersecurity over the typical 15-year lifespan of a vehicle is highly challenging due to the rapidly evolving threat landscape. The dynamic nature of cybersecurity risks means that security solutions must be adaptable, yet older vehicles may lack compatibility to support future updates.

-*Evolving standards and regulations.* Considering the dynamic threat landscape, maintaining compliance with evolving standards and ensuring secure and timely updates is a challenge for manufacturers.

1.2.1 Current trend

Cybersecurity has historically received less emphasis compared to areas such as safety, but recent regulations indicate a shift, making it a priority to align with the security and resilience requirements of automotive systems [6–8, 10, 11]. As illustrated in Figure 1.5, the industry is also increasingly adopting a more centralized architecture and Ethernet communication to meet the performance requirements of autonomous driving.

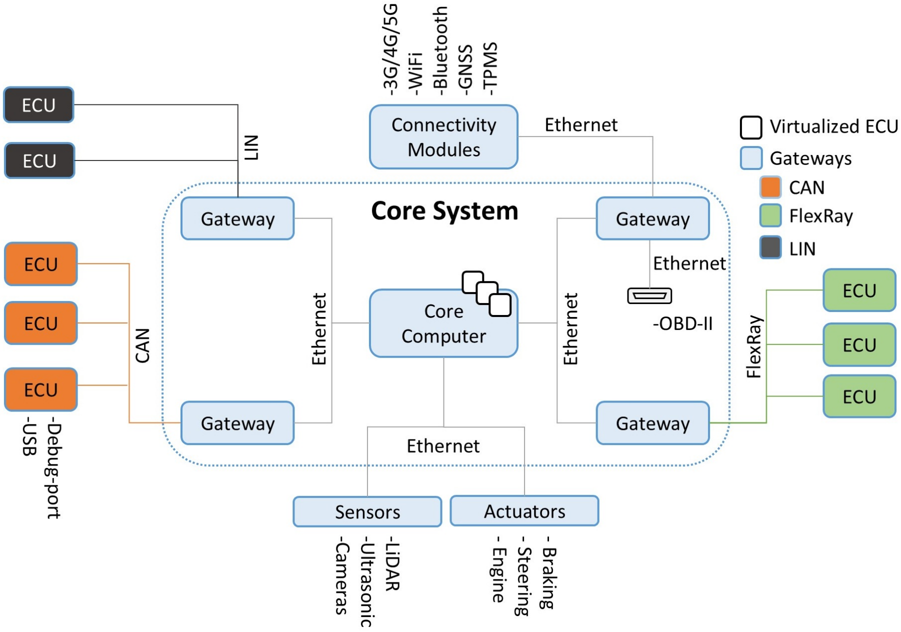


Figure 1.5: An example of a core vehicle architecture [4].

This adoption enables increased virtualization, thereby reducing the necessity of expanding architectural complexity by adding additional hardware. Thus, from a cybersecurity perspective, this transition presents an opportunity to improve the implementation of security techniques, mainly due to better performance and the potential for virtualization and isolation.

Although centralizing functionality can enhance cybersecurity, balancing this with the risk of creating a single point of failure that could be an attractive target for threat actors is essential, where, for instance, redundancy for critical functions can mitigate potential attacks.

1.2.2 Addressing challenges

In this thesis, we address many of the above mentioned challenges by aligning with regulations, guiding the development of cybersecurity and resilience mechanisms, and ensuring adaptability in proposed frameworks and architectures. For instance, a methodology was developed for identifying vulnerabilities and enhancing vehicle security throughout its lifetime, which was further utilized in a proposed resilience framework. Additionally, a versatile and secure software update reference architecture and an additional reference architecture for digital forensics are proposed, where the latter enhances fault-tracing capabilities through trustworthy data collection and analysis and improves the overall detection of vulnerabilities. These vulnerabilities can then potentially be securely patched through software updates. Thus, adopting a secure software update approach ensures alignment with an evolving threat landscape through the adaptability of software-defined vehicles. By utilizing both secure software updates and digital forensics mechanisms, synergy can be achieved, thereby enhancing the overall security of the vehicle system.

1.3 Automotive Cybersecurity, Secure Software Updates, and Automotive Digital Forensics

The primary focus of this thesis work centers around the design of safe, secure, and resilient vehicles. In the first part of the thesis, the emphasis is on the establishment of methodologies, guidelines, and principles. In the second part, the thesis focuses on architectures, introducing reference architectures for vehicle software updates and digital forensics.

This section briefly gives an overview of the main areas of the thesis: automotive cybersecurity, secure software updates, and automotive digital forensics.

1.3.1 Automotive Cybersecurity

NIST defines cybersecurity as the ability to protect or defend the use of cyberspace from cyber attacks [12], such as the protection of systems, networks, and data from unauthorized access and other malicious activities. Resilience, on the other hand, can be defined as the ability of a system to maintain its intended operation in a dependable and secure way, possibly with degraded functionality, in the presence of faults and attacks [13]. Thus, cybersecurity aims to prevent, detect, and mitigate cyberattacks, while resilience goes further by striving to endure, adapt to, and recover from cyberattacks. This ensures that systems can continue to operate effectively, regain functionality, and minimize the impact of successful attacks. Security properties, threat categorization, and potential threat actors are crucial considerations when securing vehicles and other systems. These aspects have been important to this research work.

Security properties. Cybersecurity is typically conceptualized as fulfilling security properties, with CIA (Confidentiality, Integrity, and Availability) being the most common. *Confidentiality (C)* ensures that only authorized entities can access and disclose data. *Integrity/Authenticity (I)* ensures that the information remains accurate and unaltered throughout its lifecycle. *Availability (A)* of data should be ensured to authorized entities, for example, in the event of a crash, with secure and tamper-proof storage guaranteed.

Threat Categorization. Microsoft's STRIDE categorization enables the grouping and mapping of attacks into specific attack-type categories, each connected to the security attributes the attack aims to compromise [14]. STRIDE is an abbreviation for the following, with the target security attribute in parentheses: *Spoofing (Authenticity, Freshness), Tampering (Integrity), Repu-*

diation (Non-repudiation, Freshness), Information Disclosure (Confidentiality, Privacy), Denial of Service (Availability), and Elevation of Privilege (Authorization). In addition to the CIA, STRIDE considers *Non-Repudiation (N)* that ensures that the occurrence of an event and its origin cannot be denied, where integrity and authenticity are prerequisites. Therefore, Non-Repudiation is particularly important for forensic investigations. *Privacy (P)* is related to personal data, such as traffic violations, location data, and synced data from external devices, where *Freshness (F)* ensures that the data are up-to-date and have not been replayed.

Threat Actors. The following actors aim to compromise vehicle assets based on specific motivations, i.e., the goals they seek to achieve with their attacks. *Cyber Terrorists* have ideological, political, or religious objectives. *Financial Actors*, driven by financial gain, target companies (intellectual property), organizations, or individuals. *The Foreign Country* seeks power through cyber warfare, intending to disable critical infrastructure assets (e.g., transportation). *Insiders* are motivated by retaliation or personal gain, possess sensitive knowledge, and may plant malicious code within the vehicle. *Hacktivists* are driven by the desire for publicity or the adrenaline rush, often with an agenda for political or social change. *Script Kiddies* usually have no clear objective, possess limited knowledge, and often use readily available tools and scripts.

Regulations and industry practices. From an automotive perspective, OEMs are required to align with automotive regulations. For instance, the UN R155, states that OEMs must implement appropriate cybersecurity measures in the design and shall be able to detect and respond to possible cybersecurity attacks [8]. In addition, OEMs shall provide data forensic capability to analyze attempted or successful cyberattacks. Furthermore, the UN R160 [7] states that OEMs shall store relevant data in an Event Data Recorder (EDR), and the UN R156 states that OEMs shall ensure a secure software update process [15]. To comply with UN R155 and UN R160, extensive data collection is required, which also needs to align, for instance, with the GDPR [9], which states that data should only be collected for specified, explicit, and legitimate purposes, stored for as long as necessary for the intended purposes, and processed securely to prevent unauthorized access, alteration, or disclosure.

Note that regulations outline their principles, focusing on what must be achieved without providing technical details. Regulations serve as mandatory requirements that must be fulfilled, while standards, such as ISO/SAE 21434 [6], on the other hand, offer recommendations and can be referred to when providing documentation to authorities on how regulations are met.

Security mechanisms and solutions. OEMs and third-party suppliers often develop their own customized security mechanisms and solutions. Examples are intrusion detection systems (IDS), software authentication and update solutions, secure logging, secure boot, and authentication algorithms. Formally proving the security of complex vehicle systems is challenging and not always feasible or practical. Thus, the industry typically employs a combination of methodologies to ensure system security. This approach includes following security standards and best practices, such as the ISO/SAE 21434 [6]. In addition, Threat and Risk Assessment (TARA) is a commonly employed methodology to identify and mitigate potential threats [6]. Penetration testing is another essential practice for analyzing and addressing vulnerabilities. Furthermore, standard procedures include external and internal code reviews, following secure programming guidelines like the Secure Software Development Lifecycle (SSDLC) [16], continuous monitoring, and updating throughout the vehicle's lifespan. While formal proofs can be utilized for specific functions or simplified models of complex systems to provide theoretical assurance, they are rarely applied within the industry due to their limited practical value regarding real-world systems.

Frameworks and Architectures. The Automotive Open System Architecture (AUTOSAR) [17], aims to standardize automotive software development by providing requirements and guidelines for standardized interfaces, communication protocols, and data formats. However, the number of AUTOSAR-compliant ECUs used in vehicles can vary significantly between OEMs and vehicle models, and the degree to which they comply with these standards can also differ. As discussed in Chapter 5, there is no standard for automotive digital forensics, and regarding software updates, the ISO 24089 standard [11] exists but does not provide technical details about implementation.

There is a framework specific for AUTOSAR-compliant ECUs that has specifications that developers and automotive manufacturers can use to implement their update and configuration management solutions. However, customized solutions are required for OEMs, depending on the back-end and vehicle architectures. Therefore, implementation guidelines and requirements for a secure, versatile and unified approach applicable to various architectures and use cases are needed for software updates [18, 19] and automotive digital forensics [4, 20] to guide compliance with security and forensic requirements [8, 15].

1.3.2 Secure Software Updates

Software updates are not just about adding new features or fixing vulnerabilities. Software updates also play an important role in maintaining system security. This includes updating security features such as firewall rules, configurations, and cryptographic keys, especially in response to evolving cyberthreats. The ability to apply these security updates quickly and securely is imperative. Regulations such as UN R156 [15] and the ISO 24089 standard [11] emphasize the importance of a robust and secure software update process. However, it does not provide technical details.

Updating vehicle software presents challenges due to the complex nature of vehicle architectures. ECUs operate on a diverse range of operating systems with varying performance and storage capacities. The use of different vehicle communication protocols further complicates the update process. Updating ECUs involves performing diagnostics and processing cryptographic keys [18,21]. For example, temporarily disabling firewalls and activating programming modes may be necessary for the software update process. These complexities require a thorough approach to software updates, balancing security enhancements with the complexity and dependencies on the vehicle's electrical and electronic (E/E) architecture.

1.3.3 Automotive Digital Forensics

Various vehicle cyberattacks and incidents have exposed vulnerabilities in automotive systems [22–27]. For instance, in [24], new malicious functionalities were introduced, including remote access persistence. The malicious code automatically erased any evidence of its existence after the crash. Therefore, no evidence related to a potentially life-threatening code was available. In [25], a woman was killed by an autonomous vehicle. The software did not correctly identify the individual as a pedestrian. In [26], a driver with the autopilot activated collided with a vehicle in front. Afterward, vehicle data, including sensor information, was used in digital forensic investigations. Establishing incident traceability for the driver, the autopilot, and potential threat actors is imperative.

We can assume that cyberattacks and other vehicle-related incidents will continue to rise and become even more prevalent. The increasing frequency of cyberattacks on vehicles has led to the need for aligned forensic requirements and standards. Digital forensics involves an extensive process of identifying, preserving, verifying, analyzing, documenting, and presenting digital evidence with high confidence in its admissibility, ensuring its forensic soundness [4]. However, frequently, vehicular data extraction relies on tools unsuited for digital forensics, leading to a deficiency in data forensic integrity [4]. Additionally, vehicle data storage is commonly susceptible to tampering and often lacks satisfactory security mechanisms.

The field of digital forensics within the automotive domain is relatively new, where existing self-monitoring and diagnostic systems primarily focus on safety-related events. To enable effective forensic investigations, automotive systems must be extended to securely log and store additional information, particularly regarding security relevant events.

A modern vehicle communicate internally and to the outside world via various connection interfaces, giving rise to a large amount of data, where some are imperative for digital forensics investigation. However, the vast amount of data generated by the vehicle is challenging to manage and requires automated processes. Currently, only a fraction of the available data is logged and analyzed. Data are distributed across various locations, making identification and retrieval time-consuming. Additionally, cost and performance constraints in vehicles make it challenging to implement security mechanisms that ensure trustworthy data. Finally, automotive regulations, standards, and common guidelines are limited, especially concerning forensic processes, data collection, management, formats, and tools. Therefore, standardizing automotive digital forensics is important for ensuring forensic soundness.

1.4 Thesis Objectives and Contribution

The lack of methods and techniques to ensure secure and resilient vehicle design, along with insufficient digital forensic capabilities within the automotive industry, requires urgent attention. We are rapidly heading towards a future where almost all devices, including vehicles, are interconnected in some form. This increasing connectivity and complexity increases the risk of malicious actors compromising these devices, potentially leading to hazardous consequences such as accidents. *Detecting*, *preventing*, *responding* to, as well as *predicting* potentially malicious events, are essential to ensure the safety of vehicles and their passengers. Detection is important for making informed decisions on how to handle incidents, such as whether to prevent, allow, or mitigate their impact. An effective approach to predicting future events is by understanding past occurrences. Therefore, securely tracing back previous events is vital for vehicle safety, enabling the correction of detected vulnerabilities, such as software bugs.

Additionally, identifying the origin and distinguishing between cyberattacks and hardware or software failures is important for digital forensics. Thus, a thorough understanding of existing and potential future threats, along with corresponding mitigation techniques, is necessary to develop resilient vehicles. However, research into cybersecurity, resilience, and digital forensics for modern vehicles is still in its infancy and relatively unexplored. Consequently, the core research question and main emphasis in this thesis work is:

- How can we ensure a secure and resilient vehicle design with digital forensic and secure software update capabilities?

The first part of this thesis proposes frameworks and methods to identify and mitigate vulnerabilities, ensure security and resilience, and enable traceability through digital forensic mechanisms. This sets the foundation for the second part of the thesis, which further explores secure software updates and automotive digital forensics. As shown in Table 1.1, we have defined security goals (SGs) accompanied by detailed directives on how to fulfill them (cf. Paper C). These SGs and directives establish a baseline for securing vehicles.

Table 1.1: Mapping of Security Goals to Detailed Directives

[Security Goal] [Directive]
[SG1: Secure Communication] [D1,D2] , [SG2: Readiness] [D3], [SG3: Separation of Duties] [D4, D5], [SG4:Secure Software Techniques] [D6-D10] , [SG5: Separation/Segmentation] [D11] , [SG6: Attack Detection and Mitigation] [D12-D18] , [SG7: State Awareness] [D19,D20] , [SG8: Forensics] [D1,D6,D21]
Directives
D1:Authentication, D2:Encryption, D3:Redundancy/Diversity, D4:Access Control, D5:Runtime Enforcement, D6:Secure Storage, D7:Secure Boot, D8:Secure Programming, D9: Secure Software Updates, D10:Verification & Validation, D11:Separation, D12:Specification/Anomaly-based Detection, D13:Prediction of Faults/Attacks, D14:Adaptive Response, D15:Reconfiguration, D16:Migration/Relocation, D17:Checkpoint & Rollback, D18:Rollforward Actions, D19:Self-X, D20:Robustness, D21:Forensics

The core research question is further divided into the following detailed questions specific to each paper, where Figure 1.6 provides an overview of the publications.

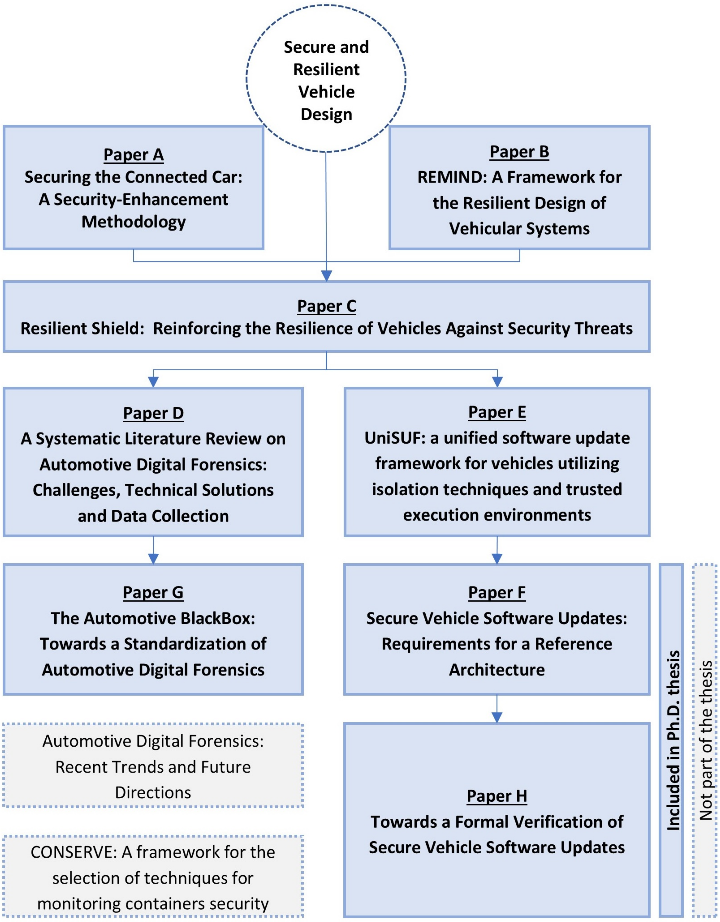

Secure and Resilient Vehicle Design
Paper A
Securing the Connected Car: A Security-Enhancement Methodology
Paper B
REMIND: A Framework for the Resilient Design of Vehicular Systems
Paper C
Resilient Shield: Reinforcing the Resilience of Vehicles Against Security Threats
Paper D
A Systematic Literature Review on Automotive Digital Forensics: Challenges, Technical Solutions and Data Collection
Paper E
UniSUF: a unified software update framework for vehicles utilizing isolation techniques and trusted execution environments
Paper G
The Automotive BlackBox: Towards a Standardization of Automotive Digital Forensics
Paper F
Secure Vehicle Software Updates: Requirements for a Reference Architecture
Automotive Digital Forensics: Recent Trends and Future Directions
CONSERVE: A framework for the selection of techniques for monitoring containers security
Paper H
Towards a Formal Verification of Secure Vehicle Software Updates
Included in Ph.D. thesis
Not part of the thesis

Figure 1.6: An overview of the publications

Part I.

- **Paper A.** What existing methods and models within the cybersecurity area apply to the automotive industry, and can these methods and models be adapted and used for vehicles?

  Outcome. Paper A introduces a systematic methodology to identify and mitigate vehicle vulnerabilities.

- **Paper B.** What work concerning resilience techniques, fault tolerance, and dependability exists in the literature, and how can these techniques be used to detect, mitigate, recover, and endure vehicle cyberattacks?

  Outcome. Paper B proposes a framework for resilience techniques, which are then incorporated into Paper C.

- **Paper C.** What types of cyber-attacks have been performed on vehicles in the last ten years, and how can we achieve security and resilience to mitigate such attacks?

Outcome. Paper C demonstrates the practical application of the methodology of Paper A and the resilience techniques from Paper B, resulting in a framework that establishes a vital baseline of protection against common security threats and attacks.

- **Paper D.** What research exists within automotive digital forensics, what technical solutions exist, and how do these maintain security properties? What forensically relevant data can be derived from existing literature, and who are the stakeholders for this data?

  Outcome. Paper D provides a comprehensive overview of the research landscape in automotive digital forensics, establishing connections between research findings and practical applications.

Secure software updates have become vital with the transition from traditional vehicles to software-defined vehicles, where software essentially controls most vehicle functions. The ability to quickly and securely patch vulnerabilities and adjust functions and security features, such as firewall rules and enabling/disabling functionality, is now imperative. We propose such an approach, the Unified Software Update Framework (UniSUF), in Papers E, F, and H.

Furthermore, given the complexities of an evolving threat landscape with increased risk of attacks targeting vehicles, digital forensics, a relatively new and uncharted territory for the automotive industry, has become important for traceability related to crime, as well as the discovery and potential mitigation of vulnerabilities. A reference architecture for automotive digital forensics is proposed in Paper G. Consequently, the second part of the thesis, comprising Papers E to H, emphasizes two security goals: SG4 - Secure Software Techniques, specifically targeting secure software updates, and SG8 - Forensics.

For instance, in Paper G, and as shown in Figure 1.7, we illustrate a mapping to ensure a secure and resilient vehicle design (cf. Table 1.1), and specifically detail the mapping of SG8 to Automotive Digital Forensics Goals (ADFG) and their corresponding requirements. For example, ADFG-1 is general and maps to *Availability and Trust*, which requires R1-R7, while ADFG-2 is more specific and relates to data *Identification*, which requires R2, R4-R6, and R9-R10.

Part II.

- **Paper E.** How can we ensure the ability to quickly and securely patch software vulnerabilities in vehicle software for various scenarios within the automotive domain?

  Outcome. Paper E introduces a framework for the rapid and secure deployment of vehicle software to address the increasing number of software bugs in vehicle systems.

- **Paper F.** What entities comprise a potential reference architecture for secure software updates, and what are the security goals and requirements needed?

  Outcome. Paper F builds upon the work presented in Paper E by introducing a reference architecture for secure software updates in vehicles. This architecture aligns with an attacker model, security goals, and requirements.

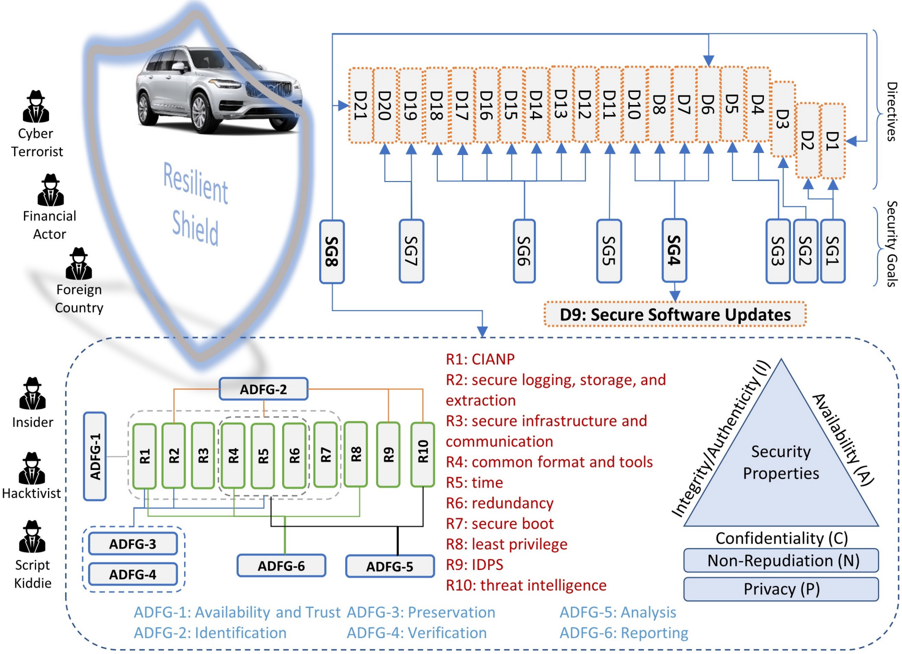


Figure 1.7: Mapping goals to the most relevant directives, which are then aligned with Automotive Digital Forensics Goals and their corresponding requirements. Additionally, the figure visualizes potential threat actors. The figure includes an image of a Volvo car [3].

- **Paper G.** What are the requirements and architectural components necessary to ensure effective digital forensic capabilities in vehicles?

  Outcome. Paper G presents a reference architecture designed specifically for automotive digital forensics.

- **Paper H.** To what extent is it possible to formally prove the security and correctness of UniSUF requirements and architecture?

  Outcome. Paper H formally proves the security and correctness of a simplified model of UniSUF.

Summary of contributions. In Paper A, we have contributed with a systematic methodology to find and mitigate vehicle vulnerabilities. In Paper C, we have shown how the methodology can be practically used to strengthen and enhance vehicle security, resilience, and safety. In Paper B, we provided a framework for resilience techniques and incorporated these techniques into Paper C. Thus, Paper A introduces the *SPMT methodology*, which is then used in Paper C to establish the *Resilient Shield*, a framework for the design of resilient vehicles. In Paper B, resilience techniques are outlined and categorized under the *REMIND* framework, laying the foundation for integrating these techniques into the detailed directives of the *Resilient Shield*. Consequently, both *SPMT* and *REMIND* serve as fundamental components for the development of *Resilient Shield*, where *Resilient Shield* outlines security goals and provides detailed directives to ensure a secure and resilient vehicle design (cf. Table 1.1). Additionally, *Resilient Shield* serves to prepare the ground for architectural design within secure software updates, namely *UniSUF* presented in Papers E, F, and H, and digital forensics in Papers D and G.

In Paper E, we provide a framework for securely deploying vehicle software to address the increasing number of software bugs and the need for dynamic adaptations in vehicle software. This framework, in conjunction with an attacker model, security goals, and requirements, has facilitated the establishment of a reference architecture for secure software updates in Paper F, with its security and correctness formally proven in Paper H. Finally, we present an extensive structured overview of the research landscape of automotive digital forensics and established connections between research findings and practical applications in Paper D, culminating in an additional reference architecture specifically designed for automotive digital forensics in Paper G. Moreover, Paper D has been added to the IEEE Standard 1616.1-2023 ANNEX A, which addresses Data Storage Systems for Automated Driving [28]. This recognition underscores the research's practical relevance and industry impact in automotive digital forensics.

Research methodology. We used a qualitative and quantitative research methodology in Papers A-D. This included conducting extensive literature reviews, critically assessing and analyzing data, and categorizing relevant information using qualitative and quantitative approaches. Additionally, we adapted and categorized these findings to suit the automotive domain's specific needs, constraints, and objectives. For Papers E-H, we employed qualitative research and engineering approaches. We integrated qualitative research to comprehend stakeholders' needs and engineering approaches to develop the solutions. We used a structured approach to formalize goals, arguments, and evidence. Our research involved defining architectures for processes, steps, and specific components, imperative for securing software update mechanisms and facilitating digital forensic processes within the automotive context. This included threat and risk assessment, requirement analysis, specification definition, architectural design, and formal proof of security and correctness.

1.5 Overview of the Included Publications

This section provides a summary of the publications in this thesis, detailing their research questions and contributions.

Paper A. Securing the Connected Car: A Security-Enhancement Methodology [29]

K. Strandberg. T. Olovsson, E. Jonsson

There is a need for approaches specifically for the automotive domain concerning security analysis, such as threat modeling, risk assessment, mitigation, and testing. However, previous approaches have been connected to areas other than automotive or limited to certain phases in a vehicle's life. Thus, in this paper, our emphasis is on the following.

- What established methodologies and models in cybersecurity apply to the automotive industry, and to what extent can they be adapted and used for vehicles?

The result is a security enhancement methodology named the *Start Predict Mitigate Test (SPMT)* for the identification and mitigation of vehicle vulnerabilities throughout their entire life cycle, from development to market. The methodology was developed by first studying various methods and models in different areas and selecting and adapting relevant parts applicable to the automotive industry. Secondly, a theoretical and empirical study of attacks was conducted to validate that vulnerabilities related to these attacks can be detected. The *SPMT methodology* was then developed based on the conclusions drawn from these two steps. The core contributions and benefits follow.

- An innovative, comprehensive, and systematic methodology has been developed by studying existing models and methods. Identified components have been connected, and new ideas suitable for vehicles have been incorporated.
- High coverage for mitigation is given against high-priority threats on an operational, safety, privacy, and financial aspect by following the proposed methodology.
- Security is approached thoroughly by analyzing both device and system levels aligned with vehicle development, production, and market phases.
- The proposed methodology is both time and cost-effective, offering adaptability to various scenarios by adapting tools and countermeasures based on the evaluated assets. Furthermore, the methodology allows for flexibility by starting in different phases to align with various development processes within the automotive industry.

In summary, a security and safety-enhancing methodology is proposed and achieved through an extensive and systematic approach to security analysis, specifically adapted for vehicles. This methodology covers security analysis for the entire vehicle life cycle, essential for the automotive industry's efforts to improve security and safety.

Statement of contributions. I contributed as the lead for the idea and performed the development of the methodology and the writing of the manuscript. The co-authors acted as reviewers and provided feedback during the process.

Appeared in: *IEEE Vehicular Technology Magazine, 2018*

Paper B. REMIND: A Framework for the Resilient Design of Automotive Systems [13]

T. Rosenstatter, K. Strandberg, R. Jolak, R. Scandariato, T. Olovsson

An evolving automotive landscape characterized by increased complexity and connectivity poses challenges in ensuring security within these systems. While the automotive industry moves towards a more centralized architecture, facilitating the implementation of security and resilience techniques, expecting protection against all cybersecurity incidents is unrealistic. Therefore, an additional approach is necessary to ensure resilience in the face of security breaches.

In essence, detection and prevention occur at the first layer. Should these measures fail, resilience to endure the breach and efforts to recover become imperative. For instance, transitioning to a secure operating state with restricted functionality can enable a safe stop for the vehicle. This paper aims to explore the following.

- What work concerning resilience techniques, fault tolerance, and dependability exists in the literature, and how can these techniques be used to detect, mitigate, recover, and endure vehicle cyberattacks?

We perform a literature study on resilience techniques, fault tolerance, and dependability. The result is the REMIND resilience framework that provides techniques for attack detection, mitigation, recovery, and resilience endurance. Moreover, we provide guidelines on how the REMIND framework can be used against common security threats and attacks, and further discuss the trade-offs when applying these guidelines. Thus, the core contributions are:

- A framework for the design of resilient automotive systems.
- Techniques are categorized into four groups: detection, mitigation, recovery, and endurance, representing their purpose.
- Techniques of different categories can be combined to create layers of security.
- Techniques are assigned to classes of automotive assets, and pros and cons are provided for each technique to further support design decisions.

In summary, we provide a framework to guide architects in selecting the appropriate resilience techniques for the design of automotive systems.

Statement of contributions. This is a joint work with the main and co-authors. Thomas Rosenstatter was the lead for the idea, the literature review, and the design of the taxonomy for resilient techniques. I have written and contributed, particularly, to the attack model, asset identification, and the associated table. Rodi Jolak was primarily responsible for the REMIND resilience guidelines.

Appeared in: *IEEE Secure Development, 2020*

Paper C. Resilient Shield: Reinforcing the Resilience of Vehicles Against Security Threats [30]

K. Strandberg, T. Rosenstatter, R. Jolak, N. Nowdehi, T. Olovsson

Various vehicle cyberattacks have occurred in the past. To investigate the risk of these attacks and ensure that future vehicles can withstand these types of attacks, we have performed a thorough threat and risk analysis of published attacks on vehicles from the past ten years. Thus, in this work, we investigate the following.

- What types of cyberattacks have been performed on vehicles for the last ten years and how can we achieve security and resilience to mitigate such attacks?

We have performed the *SPMT methodology* proposed in Paper A for the complete vehicle as produced by the manufacturer. In the first phase of the *SPMT*, the *Start Phase*, we define vehicle assets, threat actors, their motivations, and objectives, which gives rise to a threat model. In the second phase, the *Predict Phase*, we study attacks that could affect the defined assets from the previous phase. We analyze these attacks based on attack probability and consequence criteria and assign a risk value for each attack. Each attack is further categorized into a category based on the attack type. This phase gives rise to an attack model. In the next phase, the *Mitigate Phase*, we define required security and resilience enhancements against all threats as security goals. In addition, we define detailed directives on how to fulfill these security goals. The resilience techniques defined in Paper B are further incorporated into these detailed directives. The *Mitigate Phase* gives rise to the *Resilient Shield*, a framework for designing resilient automotive systems. Thus, the core contributions are:

- By applying the *SPMT methodology*, we performed an extensive threat and risk analysis of 52 published attacks on vehicles in the past ten years.

- We developed a threat model to secure vehicles by identifying vital vehicle assets and related potential threat actors, their motivations, and objectives.

- We developed an extensive attack model created from the analysis of the identified threats and attacks, further filtered and categorized based on attack type and risk criteria related to the probability and consequences of the attack.

- We present an exhaustive mapping between asset, attack, threat actor, threat category, and resilience mechanism for each attack.

- We define necessary security and resilience enhancements for vehicles, validating the effectiveness of the *SPMT methodology*.

In summary, we propose *Resilient Shield*, a comprehensive resilience framework that considers actual cyberattacks against defined automotive assets. Mitigation techniques are proposed, giving rise to a vital baseline of protection against common security threats and attacks.

Statement of contributions. I contributed as the lead for the idea and performed the writing of the manuscript. I also contributed to the identification of the assets, threat actors and their motivations and objectives, and the high-level security goals. The identification of the detailed directives and the work to find and review attacks has been a joint effort between me and the secondary author. The assignment and mapping of mitigation techniques, STRIDE categories, automotive assets, and threat actors have been a collaborative effort between the authors.

Appeared in: *IEEE 93rd Vehicular Technology Conference, 2021*

Paper D. A Systematic Literature Review on Automotive Digital Forensics: Challenges, Technical Solutions and Data Collection [4]

K. Strandberg, N. Nowdehi, T. Olovsson

The modern vehicle landscape is characterized by complex internal architectures and wireless connectivity to the Internet, other vehicles, and infrastructure. Connectivity, while offering numerous benefits such as convenience and efficiency, also introduces significant cybersecurity challenges. For instance, increasing the risk of cyberattacks and other criminal incidents.

Recent incidents involving autonomous vehicles stress the need for more research into Automotive Digital Forensics (ADF). Failures in automated driving functions can originate from various sources, including hardware and software malfunctions, as well as cybersecurity vulnerabilities. Thus, it becomes vital to determine and analyze the root causes of these failures, a task that relies on trustworthy data.

ADF represents a relatively new domain within the automotive industry. Currently, most existing self-monitoring and diagnostic systems in vehicles primarily focus on safety-related events. Although identifying the root causes of failures is essential for traceability for digital forensics and for improving the safety and reliability of vehicles, limited work has been done in this field.

In this paper, we put effort into the identified *SG8: Forensics* from Paper C and perform an extensive review of the area of automotive digital forensics. To the best of our knowledge, our study represents the first systematic literature review within the domain of automotive digital forensics. We analyze more than 300 papers published between 2006 and 2021, offering a thorough assessment of the current research landscape in this field.

Our approach involves categorizing the identified literature into distinct focus areas, providing a structured overview of the diverse topics and research activities within automotive forensics. Furthermore, we go beyond identification and assessment by linking forensically relevant data extracted from the literature to specific categories. This process allows us to map the data to essential security properties and potential stakeholders, offering valuable insights for practitioners and researchers.

In summary, our systematic categorization enables quick access to relevant work within specific sub-fields of ADF, simplifying the process of navigating the extensive literature base. By providing a structured overview of the research landscape and establishing connections between research findings and practical applications, our work serves as a valuable resource for guiding future research activities and the development of forensic mechanisms by engineers.

Statement of contributions. I have performed the database searches and investigated each database's individual coverage and specificity. I performed backward and forward snowballing on the selected work to increase the coverage. I identified the categories based on focus areas in the selected work and mapped the technical solutions to the considered security properties. Additionally, I identified and categorized forensically relevant data and mapped this data to potential stakeholders. I also identified and discussed challenges, issues, and research gaps within the area of automotive digital forensics. I performed the writing of the manuscript.

Appeared in: *IEEE Transactions on Intelligent Vehicles, 2023*

Paper E. UniSUF: a unified software update framework for vehicles utilizing isolation techniques and trusted execution environments [31]

K. Strandberg, D. K. Oka, T. Olovsson

Software plays a vital role in vehicles, with code bases often exceeding 100M lines. Software controls various functionalities, such as braking, steering, and engine control. Software code is anticipated to expand to roughly 300M lines within the next decade, and despite rigorous testing, there's predicted to be around one bug for every 1000 lines of code [32]. Thus, for modern vehicles, this indicates approximately 100,000 bugs, which may increase to around 300,000 within a few years. The severity of vulnerabilities in vehicle software can range from minor disruptions, such as blinking lamps, to critical malfunctions in safety systems, like braking or steering, potentially leading to hazardous consequences. Therefore, the capability to address these vulnerabilities promptly and securely is vital and serves as a fundamental requirement in securing modern vehicles. However, existing solutions often lack the necessary details for a versatile, unified, and secure approach covering various update scenarios, including over-the-air updates, workshop installations, factory production updates, or the use of diagnostic update tools. Thus, in this paper, we investigate:

- How can we ensure the ability to quickly and securely patch software vulnerabilities in vehicle software for various required scenarios within the automotive domain?

Consequently, one of the stated security goals from Paper C is investigated more in-depth: *SG4: Secure Software Techniques.* The core contributions are:

- We have investigated several software update use cases and identified constraints and conditions for a unified and versatile approach.
- Considering these constraints and conditions, we propose an approach for vehicle software updates, where all data needed for a complete software update is securely encapsulated into one single file.
- This file can be processed in several update scenarios and executed without any external connectivity since all data is inherently secured.
- We provide an extensive overview of a possible secure implementation covering the whole software chain from producer to receiver.
- Our approach has been reviewed with automotive software update architects to ensure that the proposed approach can be practically deployed and efficiently adopted for vehicle software updates.

In summary, we propose the *Unified Software Update Framework for Vehicles (UniSUF)* that utilizes isolation techniques and trusted execution environments to ensure a secure software update process.

Statement of contributions. I contributed as the lead for the idea, performed the development of the framework, and the writing of the manuscript. The co-authors acted as reviewers and provided feedback during the process.

Appeared in: *19th escar Europe: Embedded Security in Cars, 2021*

Paper F. Secure Vehicle Software Updates: Requirements for a Reference Architecture [19]

K. Strandberg, U. Arnljung, T. Olovsson, D. K. Oka

Securing the software update process for vehicles, especially in the face of insecure protocols and ECUs, presents significant challenges. Addressing this issue demands an adaptable update framework capable of accommodating diverse architectures. UniSUF proposes an add-on framework designed to meet these challenges by operating as a first layer of defense. This approach ensures compatibility with existing and forthcoming architectures, where UniSUF preserves the ability for each ECU to conduct its own software verification, such as secure boot and authentication of software and communication. By offering this level of autonomy and security, UniSUF enhances the overall resilience of vehicle systems against cyber threats while accommodating the complexities of varied ECU architectures.

In this paper, we continue our work within *SG4: Secure Software Techniques*, by outlining an attacker model for UniSUF along with general security requirements, further connected to common security goals and directives to ensure extensive coverage. We identify entities involved during vehicle software updates, perform a threat assessment, and map the identified threats to security goals and requirements, resulting in a reference architecture with high industrial relevance, applicable to guide not just to automotive but also related areas such as cyber-physical systems, internet-of-things, and smart cities.

Statement of contributions. I contributed as the lead for the idea, performed the analysis, the identification of security goals, directives, and security requirements, as well as the writing of the manuscript. The co-authors acted as reviewers and provided feedback during the process.

Appeared in: *IEEE Vehicular Technology Conference, 2023*

Paper G. The Automotive BlackBox: Towards a Standardization of Automotive Digital Forensics [20]

K. Strandberg, U. Arnljung, T. Olovsson, D. K. Oka

Digital forensics involves the collection, preservation, analysis, and presentation of digital evidence to investigate incidents and determine the sequence of events that lead to a particular outcome. In the automotive domain, digital forensics aims to provide insight into accidents, system malfunctions, or cyberattacks involving vehicles.

Our contributions include a threat model inspired by our work in Paper C, where six threat actors are identified. We assume that they mutually aim to execute various cybercrimes aimed at vehicles, potentially impacting the driver, passengers, and surrounding objects by exploiting the vehicle itself. Nevertheless, the primary goal remains concealing, erasing, or altering digital evidence, including traces of criminal activities, to impede or halt forensic investigations. Considering the aforementioned threat actors and their agenda, we define six Automotive Digital Forensics Goals (ADFG). We also identify specific ADF requirements and map them to the six stated ADFGs. Considering the threat model, we propose a reference architecture for automotive digital forensics that meets the specified ADFGs and requirements. This architecture can serve as a blueprint for designing digital forensic-enabled vehicles.

Statement of contributions. I contributed as the lead for the idea, the identification of the forensic goals and requirements, and their mapping. I developed the reference architecture and also wrote the manuscript. The co-authors acted as reviewers and provided feedback during the process.

Appeared in: *IEEE International Workshop on Information Forensics and Security, 2023*

Paper H. Towards a Formal Verification of Secure Vehicle Software Updates

M. S. Hagen, E. Lundqvist, A. Phu, Y. Wang, K. Strandberg, E. M. Schiller

Formal proofs are seldom used in the automotive industry for complex systems. However, it can be applied to simplified models. We have broken down UniSUF into smaller subsystems and made simplifications and assumptions where necessary. Each subsystem is proven individually and, when connected, also for the entire system. The following security properties and goals are derived from UniSUF. *Confidentiality.* Ensure the confidentiality of software during updates and that cryptographic keys are only decrypted and processed by authorized entities. *Integrity and Authenticity.* Verify that the software remains authentic and unchanged during updates and that only authentic resources are processed. *Freshness.* Prevent adversaries from reverting software to older versions and ensure that each Vehicle Unique Update Package (VUPP) is used only for its specific vehicle. *Order.* The software update process should be performed in the correct order. *Liveness.* Ensure that the software update process always terminates.

This paper addresses the following research questions by proving that UniSUF satisfies the specified security properties and goals.

- Can we ensure that critical secrets, such as cryptographic material, are not exposed by any processes?
- Can we verify that the distributed software comes from an authentic source and remains unaltered?
- Can we prevent downgrading to previous software versions?
- Can we ensure that the update process follows the correct order?
- Can we guarantee that the update process always terminates?

The complete symbolic execution of processes within UniSUF utilizing the tool ProVerif shows that all stated goals and requirements are fulfilled and, therefore, are formally proven for security and correctness. Our formal verification provides a strong foundation, but does not fully ensure the security of a real-world implementation of UniSUF. Thus, additional research is needed to align formal models with real-world applications.

Statement of contributions. This work is a collaborative effort involving four MSc thesis students engaged in an industrial MSc project, with E. M. Schiller and myself acting as supervisors. My role focused on defining and aligning architectural requirements, as well as providing continuous input throughout the project.

Under submission.

1.6 Future Work

Paper A considers various vehicle phases, including development, production, and market phases. However, it does not specifically address the vehicle's end-of-life, including the management of sensitive data contained within the vehicle after its decommissioning (e.g., data privacy considerations). While the focus of Paper A is on the phases where security concerns are more prominent, which justifies not covering the end-of-life phase in detail, there is an opportunity to expand the research to include this phase in future work. Furthermore, the proposed structured approach, designed to be easily repeated for vehicle changes, does save time; however, such processes are still inherently time-consuming. Future work could explore automation through emerging technologies, such as Artificial Intelligence (AI) and Machine Learning (ML), to reduce time consumption and further improve efficiency.

The techniques presented in Paper B are highly relevant for securing vehicles. While the pros and cons are considered, future work could consider a deeper analysis of challenges such as architectural complexity and cost implications to enhance the practicality and real-world applicability of such techniques. Paper C focuses on cyberattacks that occurred over a 10-year time span. Although these attacks are highly relevant, it is important to consider how the framework can potentially evolve, particularly in the context of emerging threats, such as AI-driven attacks. Thus, while the framework offers a solid foundation, its applicability in the face of rapidly evolving threats can be further evaluated. For instance, the data generated when the framework in Paper C is applied to various assets could be used to create data models for ML. These data models, along with other data sources such as dynamic Software Bill of Materials (SBOMs) [33] and Common Vulnerabilities and Exposures (CVE) databases, could improve the efficiency of vulnerability detection and enable a more automated and proactive approach. Paper D is based on a fixed time period, while the environment itself is highly dynamic and quickly evolving. Although not included in this thesis, a paper currently under submission extends the work to also cover the period from 2021 to early 2025. However, a key consideration is that state-of-the-art research evolves rapidly, which is something to consider for future research.

Regarding Papers E, F, and H, UniSUF, with its reference architecture, is designed to be dynamic and compatible with most setups, as it operates on top of existing vehicle architectures. A proposal for future work could involve implementing the various entities, possibly through virtual entities (e.g., virtual machines (VMs) and containers), as a proof of concept (PoC). As previously mentioned (cf. Section 1.3.1), proving the correctness and security of real-world implementations of complex systems is challenging. Therefore, proofs are typically provided for simplified models of real-world representations. Thus, the next step could be to implement a simplified UniSUF model as a PoC, using Papers E, F, and H as a guide. Finally, Paper G, with the Automotive Blackbox architecture, is not built on top of existing architectures, as proposed in UniSUF. Instead, it requires adaptations to current architectures or the development of entirely new ones, which presents challenges, particularly regarding cost and vehicle complexity. Future work could further explore these challenges.

1.7 Summary and Conclusion

We have presented an overview of the automotive industry's transition from its original function solely as a means of transportation to its current state as a highly advanced computerized system. This evolution highlights its emergence as a complex cyberphysical entity. We have detailed the complexity of vehicle architectures, including communication interfaces and interconnections with other vehicles and infrastructure. We have highlighted the unique challenges of automotive cybersecurity and digital forensics, along with our contributions to these fields.

This thesis work is organized into two distinct categories. Papers A to D offer frameworks and guiding principles for automotive cybersecurity, resilience, and digital forensics, while Papers E to H specifically guide the design of architectures for secure software updates and digital forensics. The first part lays the ground for the second part of the thesis by identifying security goals and detailed directives, creating a baseline for securing vehicles. The second part of the thesis emphasizes secure software updates and digital forensics solutions and architectures. This focus arises from the transition from traditional vehicles to software-defined vehicles, where software controls most vehicle functions. The potential to securely patch vulnerabilities and adjust functions and security features is imperative for various use cases, such as in a factory, workshop, or over-the-air updates. Furthermore, due to the complexities of an evolving threat landscape and the increased risk of attacks targeting vehicles, digital forensics has become essential for crime traceability and the discovery and potential mitigation of vulnerabilities.

In summary, this thesis presents tools and methodologies to enhance cybersecurity within the automotive domain. It proposes a proactive strategy to anticipate and address vehicle vulnerabilities, including integrating resilience techniques into vehicle design. In addition, it provides valuable information and guidelines for automotive digital forensics. It also offers two reference architectures: one for secure vehicle software deployment and another for automotive digital forensics. These contributions not only improve cybersecurity practices in the automotive industry but also have the potential to inform and contribute to future standardization efforts and upcoming regulations in this field. Notably, automotive digital forensics is a relatively novel area with almost no information concerning standardization or regulations. Consequently, more work is needed to fully understand the necessary mechanisms to achieve a forensically enabled vehicle that aligns with the dynamic and evolving threat landscape. Similarly, the same holds true for vehicle software updates. More research is required to fully comprehend the complexities of achieving rapid and secure adaptability of software-defined vehicles. This includes additional architectural work to ensure secure end-to-end adaptation, considering infrastructural aspects where vehicles are integrated into smart infrastructure, and aligning formal models with real-world implementations.

Chapter 2

Securing the Connected Car: A Security Enhancement Methodology

Format-adapted version that appeared in IEEE Vehicular Technology Magazine 2018

K. Strandberg, T. Olovsson, E. Jonsson

Abstract. A new era is upon us, an era where Internet connectivity is available everywhere and at all times. Cars have become very complex computer systems with about 100 million lines of code and more than 100 electronic control units (ECUs) interconnected to control everything, including steering, acceleration, brakes, and other safety-critical systems. However, cars were never created with Internet connectivity in mind, and adding this connectivity as an afterthought raises many security concerns. As a result, a security-enhancing approach that considers the entire process from product development to market introduction is required.

This article suggests using a methodology known as start, predict, mitigate, and test (SPMT). Its purpose is to predict and mitigate vulnerabilities in vehicles using a systematic approach for security analysis specifically adapted for vehicles. The SPMT methodology builds on existing methodologies and models that are applicable to different phases in a vehicle's life cycle as well as on new ideas. Unlike other methods, however, the SPMT methodology covers a vehicle's entire life cycle, which results in security and safety enhancements, something that cannot be achieved by existing methodologies.

2.1 Introduction

The Internet of Things (IoT) has moved society into a new era where a growing number of devices have Internet capabilities and are behaving more like computers (e.g., smart TVs and washing machines). Access possibilities are provided via USB sticks, Bluetooth devices, or Wi-Fi/cellular connections. Modern cars can have more than 100 ECUs and contain roughly 100 million lines of code [23]. Today, a car is not just a car, it is a computer on wheels. ECUs are responsible for various safety functions such as steering and brakes, and new functionality is constantly being introduced in the automotive industry, which calls for an increase in the number of ECUs and the amount of code. As a result, this increases the likelihood of attacks by hackers. Electrical systems in vehicles are no longer isolated systems, but, rather, they are vulnerable to cyber-attacks.

2.1.1 Context

A vehicle is a safety-critical system, which means that vulnerabilities can potentially lead to life-threatening hazards. Koscher et al. [24] discuss security implications in vehicles, such as vulnerabilities in the controller area network (CAN) protocol. CAN nodes communicate with broadcast messages, which enables the possibilities for malicious devices to detect, manipulate, and inject messages anywhere on the CAN bus. Furthermore, CAN is a priority-based protocol with no authenticator field and is therefore vulnerable to both denial of service (DoS) attacks and spoofed packets. Access control, which enables firmware updates, is usually performed using obsolete proprietary encryption algorithms. In many cases, these algorithms are publicly known [24] and use short cryptographic keys indicating that the access control is vulnerable to various kinds of attacks (e.g., a bruteforce attack followed by malicious code injection through a firmware update). Koescher et al. managed to disable CAN communications and update the firmware while a test subject was driving. They have also found imperfect network segmentation, thereby using the infotainment unit in the vehicle as a bridge to attack other vehicle ECUs. Furthermore, they found that reverse engineering of the firmware is usually not necessary for attacks, since a relatively small range of CAN packets is valid. This makes it possible to simply “fuzz” the network with various packets and observe the response.

By exploiting these and other vulnerabilities, the commandeering of many vehicle functions, including safety-critical functions both in driving mode at high speed and while at a standstill, is an unfortunate reality. Considerable vulnerability is demonstrated in vehicles when physical access is obtained. However, the Koscher et al. research was met with resistance from the automotive industry for not being relevant; the industry argued that, with physical access, an individual could just as well cut cables or destroy other components in the vehicle. In response, Checkoway et al. [23] provided evidence that external attacks are also possible via a wide range of entry points and categorized these attacks into three groupings: *indirect physical access*, *short-range wireless access*, and *long-range wireless access.*

2.1.2 Indirect Physical Access

Indirect physical access can refer to the vehicle's media player, since music can be crafted with malicious content. The media player can also be used to bridge an attack on other components in the vehicle if it is not adequately isolated from the vehicle's main network. This was demonstrated when a standard International Organization for Standardization 9660-formatted compact disk that contained a specific filename was inserted into a vehicle's media player, and the content of the file was automatically used to re-flash the unit [23]. Additionally, since the media player can parse complex files, they managed to add content to Windows Media audio files, that played normally on a PC, but, in a vehicle, had the side-effect of also sending arbitrary CAN packets on the vehicle network.

2.1.3 Short-range Wireless Access

Short-range wireless access refers to Bluetooth devices, remote keyless entry, and the tire pressure monitor system. Checkoway et al. demonstrated successful attempts to compromise a vulnerable Bluetooth implementation in a telematics unit. They planted a Trojan horse in an application for an Android 2.1 operating system that could be uploaded to the Android market. For example, when a device that contains this application is paired with the vehicle, it would exploit a buffer-overflow vulnerability enabling the execution of arbitrary code in the telematics unit. Checkoway et al. also successfully paired a laptop to the vehicle with no user interaction.

2.1.4 Long-range Wireless Access

Long-range wireless access refers to global positioning systems, digital audio broadcasting, and remote telematic systems. Checkoway and his colleagues used a cellular connection to first exploit an authentication vulnerability, and then a buffer-overflow vulnerability in the telematics unit. Following this, the telematic unit was forced to download additional malicious content from the Internet. Moreover, for each vulnerability they demonstrated, they were able to obtain complete control over the vehicle's systems. Remote attacks received considerable attention when Charlie Miller and Chris Valasek performed a successful attack on an automobile by using the Internet to gain control of its vital systems [34]. Similar to Checkoway et al.'s experiments, Miller and Valasek used a cellular channel in the vehicle as leverage, but, in this case, an open port (6667) made this attack possible. As a result, 1.4 million vehicles were recalled by the manufacturer. Another attack occurred when Samy Kamkar managed to remotely unlock an OnStar-enabled General Motors car [35].

2.1.5 Goal and Approach

The goal of this article is to introduce a practical, security-enhancing methodology for identifying and mitigating vulnerabilities in vehicles throughout their life cycle (i.e., from the product development phase to the market introduction phase). The approach used for creating the SPMT methodology was divided into three parts. First, a study of various security methods and models used in different areas was conducted, and methods or parts of methods relevant to the automotive industry were selected. Secondly, a theoretical and empirical study of attacks against vehicles that have occurred was conducted. The understanding of how and why attacks are successful is necessary to develop a methodology capable of finding vulnerabilities related to these attacks, as well as to help identify other vulnerabilities. Lastly, the methodology was defined and implemented based on the conclusions from the two previous parts.

2.2 Models, concepts, and tools

The definition of a security model varies depending on the category. Generally, it specifies how to enforce a security policy and defines how to maintain security for a system or entity. The following models in particular have been effective as they pertain to the development of the SPMT methodology.

- *Spoofing of user identity tampering repudiation information disclosure DoS elevation of privilege (STRIDE).* A threat model proposed by Microsoft as a scheme for categorizing and identifying known threats according to different vulnerabilities or the malicious intents of the attacker [14].

- *Damage reproducibility exploitability affected users discoverability (DREAD).* A model also proposed by Microsoft as the next step after threat modeling. It is used to evaluate the risk for each threat by quantifying, comparing, and prioritizing the risk [14].

- *E-safety vehicle intrusion protected applications (EVITA).* A European Union project whose objective was to provide the basis for secure release of applications based on vehicle-to-infrastructure and vehicle-to-vehicle communication. EVITA proposed a method for security and safety/risk analysis for a vehicle's network and also proposed a secure architecture and communication protocol [36].

- *Healing vulnerabilities to enhance software security and safety (HEAVENS).* A project that was conducted between 2013 and 2016 [37]. The HEAVENS security model policy is divided into two categories: security objectives and security attributes. The security objectives are taken from the EVITA model, and the security attributes are taken from the STRIDE model. The main workflow from top to bottom for HEAVENS is shown in Table 2.1.

Table 2.1: The workflow of HEAVENS

TOE	Define Target of Evaluation
Threat Analysis	Define the possible threats against the TOE
Risk Assessment	Grade the severity of those threats
Security Requirements	Define needed mitigations against those threats

- *Threat agent risk assessment (TARA)*: a predictive methodology developed by Intel to define the security risks that are most likely to occur. This methodology is based on the presumption that it is too expensive and impractical to defend against all possible vulnerabilities; therefore, by choosing the most important ones, results are maximized and costs are minimized. TARA identifies all possible threats by assessing different lists of known threats. Those threats are then filtered so that only the most serious ones remain [38].

The following concepts and tools are used in this article:

- *Target of Evaluation (TOE).* This is the product or system that is the subject of evaluation.
- *Attack Tree.* A diagram visualizing the steps needed for a realized threat on an asset.
- *Threat Modelling.* The first step when creating a threat model is to evaluate which assets need protection and how they are threatened. Known vulnerabilities and attacks (i.e., searching known vulnerability databases) are then identified.
- *Risk Assessment.* This is a continuation of the previous step when the risks of the threats are evaluated. The threats are prioritized with respect to the probability of occurrence and its consequences; these are weighted against the cost of mitigation. After a threat modeling and a risk assessment are performed, it is important to evaluate the possible mitigations.
- *Attack Attempts.* A thorough understanding of attacks related to the TOE and its corresponding hacker tools is imperative. Hence, studies of attacks and adaptations of these attacks as tests should be a part of a security evaluation verifying mitigations. To verify this concept we performed attacks against the vehicle Wi-Fi network and the Volvo On Call service in a Volvo XC90 as shown in Figure 2.1 [39].

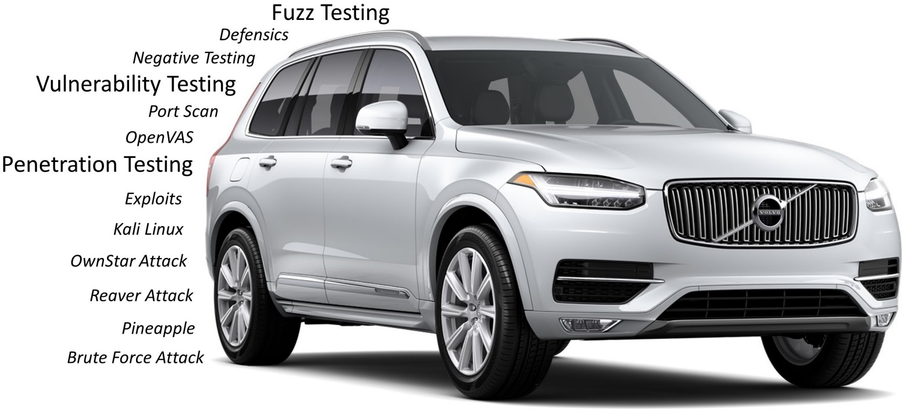


Figure 2.1: Attack attempts against Volvo XC90

2.3 The SPMT Methodology

The SPMT methodology is divided into two main stages. First, the procedure is applied to a specific TOE, i.e., an ECU or other unit. The TOE in question is then integrated into the vehicle and becomes a part of the vehicle's network. The procedure will then be repeated and adapted to the integration and performed again. By considering the devices, the integration, and system tests as a whole, a broad security perspective is achieved.

Figure 2.2, shows the procedures and concepts that have inspired the development of the SPMT methodology. The start phase defines the TOE, security policies, and a brief threat modeling based on common security and safety concepts. The predict phase consists of threat modeling by using the STRIDE threat model for categorizing vulnerabilities and threats in different lists. The idea of filtering comes from the TARA methodology (although the filtering of these lists is based on a qualitative assessment inspired by the DREAD methodology), the EVITA, and the HEAVENS projects, and by the establishment and analysis of attack trees.

The mitigate phase is based on brainstorming and further analysis of countermeasures related to the attack trees from the predict phase, as well as a quantitative assessment. Hence, a hybrid approach of a qualitative and quantitative assessment is considered. The test phase consists of practical security-related tests in conjunction with automated fuzz and vulnerability scanning tools. Penetration testing inspired by hacker attacks and their tools results in the creation of tests and tools that are adapted for the TOE. Figure 2.3 shows an illustration of the input and output to each phase.

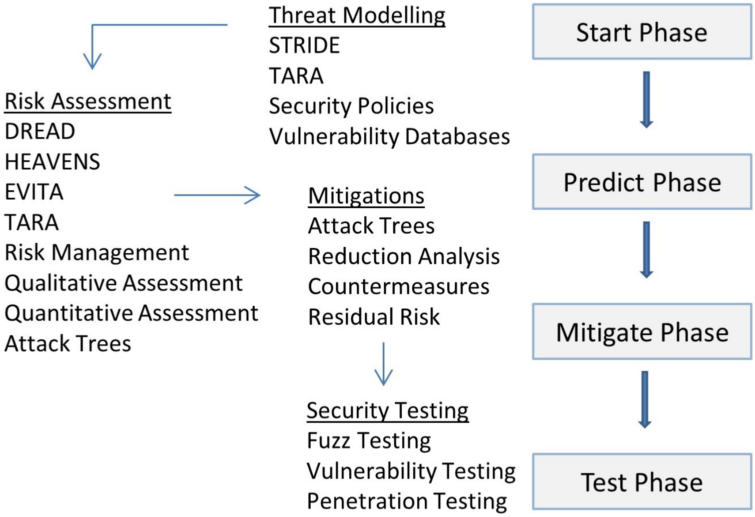


Figure 2.2: Illustration of the SPMT phases

2.3.1 Start Phase

The start phase is conducted in a workshop-like manner with discussion and brainstorming sessions on how to best implement this idea. This phase is divided into two stages:

1. The establishment of what needs protection, i.e., what is the TOE? Which assets are affected? How are those assets interconnected? How can a

compromised asset affect other assets? What are the possible entry points for threats to the vehicle?

2. Defining security policies, i.e., what does it mean for the system to be secure/insecure? How are security and safety-related attributes achieved for the TOE, such as those listed in Table 2.2?

Table 2.2: Security and Safety attributes

Confidentiality	Authorization	Privacy	Reliability
Integrity	Authenticity	Isolation	Least privilege
Availability	Freshness	Maintainability	

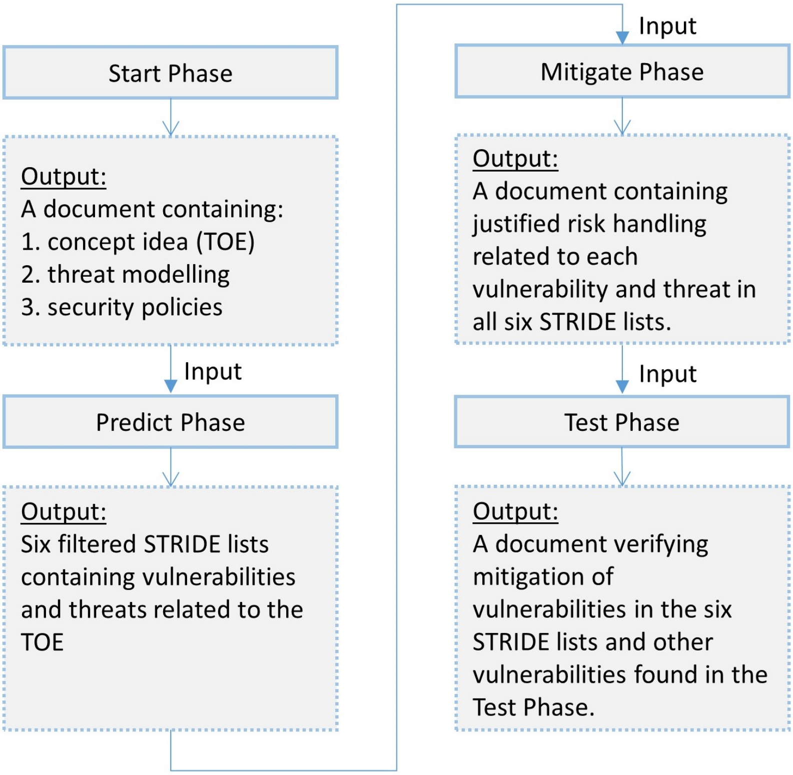

Start Phase
Output:
A document containing:
1. concept idea (TOE)
2. threat modelling
3. security policies
Input
Predict Phase
Output:
Six filtered STRIDE lists containing vulnerabilities and threats related to the TOE
Input
Mitigate Phase
Output:
A document containing justified risk handling related to each vulnerability and threat in all six STRIDE lists.
Input
Test Phase
Output:
A document verifying mitigation of vulnerabilities in the six STRIDE lists and other vulnerabilities found in the Test Phase.

Figure 2.3: Illustration of the input and output to SPMT phases

This phase produces a document containing the concept idea, a brief threat model for the TOE, and security policies, including high-level directives for the enforcement of security attributes.

2.3.2 Predict Phase

This phase consists of predicting threats against potential vulnerabilities in the TOE and its related assets. This is partly accomplished by searching known vulnerability databases, e.g., the common vulnerability enumeration (CVE) database.

Table 2.3: The six STRIDE lists

Spoofing	Authenticity/Freshness	S-LIST
Tampering	Integrity	T-LIST
Repudiation	Non-Repudiation/Freshness	R-LIST
Information disclosure	Confidentiality/Privacy	I-LIST
Denial of Service	Availability	D-LIST
Elevation of Privilege	Authorization	E-LIST

As shown in Table 2.3, this list is filtered by keywords and divided into six lists based on the STRIDE threat model. It is then managed by applying four steps:

1. Compose six lists based on the STRIDE model centered on the CVE identifier.

2. Automate the filtering of these lists based on keywords, i.e., excluding threats not relevant to the TOE.

3. Evaluate the vulnerabilities in each list, and remove those that are not relevant (the filtering from the previous step makes possible a more manually filtered approach).

4. Perform a qualitative assessment of the remaining vulnerabilities based on the calculated risk value (i.e., risk = probability for a realized threat x consequences):

   - Grade the probability of a realized threat on a scale from 1 to 3 (i.e., 1 = low, 2 = middle, 3 = high). Base the probability for a realized threat on how easy it is to exploit a vulnerability in the following manner:
     - Where, when, and in what situation can the attack be carried out?
     - What expertise is required of the attacker? What tools are needed and how difficult is it to acquire these tools?
     - What time is needed to perform the attack?

     Based on the answers to these questions, the probability of a realized threat is graded and the highest value (low, middle, or high) is chosen.

   - Grade the consequences on a scale from 1 to 3 (i.e., 1 = low, 2 = middle, 3 = high). If the attack is successful, what are the consequences? The consequences of the following four attributes (the objectives from EVITA) are graded and the highest value (low, middle, or high) is chosen.
     - Operational (graded on a scale from 1 to 3): Are any operational factors affected?
     - Safety (graded on a scale from 1 to 3): Is safety affected?

  - Privacy (graded on a scale from 1 to 3): Is any personal information compromised?
  - Financial (graded on a scale from 1 to 3): How are financial factors affected?
- Calculate the risk by multiplying the result from the probability for a realized threat with its corresponding consequences, as shown in Figure 2.4. These values determine the risk severity graded on a scale from 1 to 9.

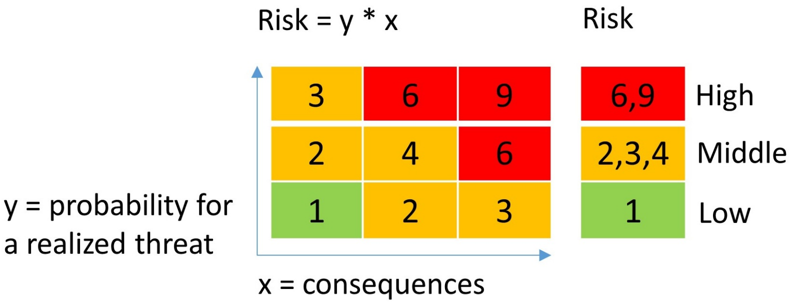


Figure 2.4: Visualizing the risk calculation process

- Remove all vulnerabilities that result in a risk = 1 from the list (acceptable risk).
- Add other relevant threats which are missing from each category by brainstorming. Consider both inside and outside threats, i.e., employees with privileges and attackers without privileges.
- Sort the remaining threats in the lists based on risk value, with the highest value first.
- Create attack trees for the threats with a risk value of 6 or 9 (prioritized threats) to connect the vulnerability with the threat, i.e., the conditions (leaf nodes) that need to be met for a successful attack (i.e., an attacker traverses from a leaf node to the root of the tree).
- Assess these vulnerabilities and threats in the next phase.

This phase produces a document containing six filtered STRIDE lists containing vulnerabilities and threats to the TOE. Note that each TOE needs its own set of lists. This requires considerable effort initially, but, once it is done, it only needs to be assessed when new potential vulnerabilities or threats are discovered.

2.3.3 Mitigate Phase

The filtered STRIDE lists are assessed from the previous phase in a workshop-like manner through brainstorming. The vulnerabilities with the highest risk are considered first, and mitigation techniques that prevent attacks are discussed and implemented if possible. This is done by analyzing the conditions (leaf nodes) in the attack trees from the predict phase, e.g., by placing a countermeasure at each leaf node. The closer to the root a countermeasure is situated, the more leaf nodes are covered. Some leaf nodes can be attained by more than one attack; hence, a countermeasure can mitigate more than one attack. Figure 2.5 shows an example of an attack tree visualizing different attacks affecting the brakes in a vehicle. Countermeasures at the node labeled Tamper with the ECU software provide protection against all three attacks in the attack tree. We can also see that we need other countermeasures relating to the node labeled DoS attack. Finding these commonalities is termed a reduction analysis and can be very effective.

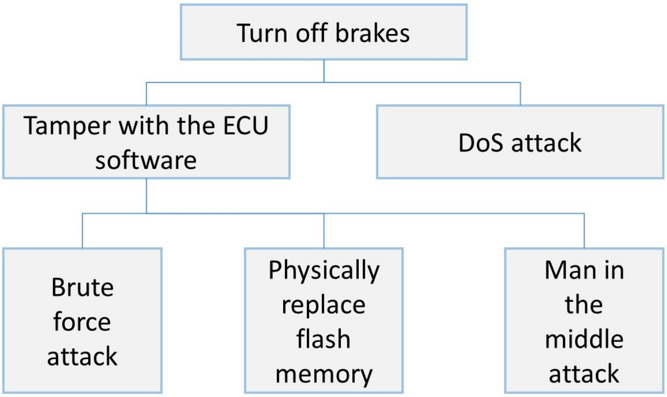


Figure 2.5: A simplified example of an attack tree

In this process, the cost for mitigation versus the value of the asset (TOE) is considered (since this enables cost-efficient mechanisms). This value is estimated by a quantitative assessment calculation, i.e., Annualized Loss Expectancy (ALE) = Annual Rate of Occurrence (ARO) x Single Loss Expectancy (SLE). In turn, the SLE is the product of the asset value (AV) and the exposure factor (EF). The exposure factor describes the monetary asset loss for a realized threat expressed in a percentage [40].

For example, if AV equals 1,000 units and EF equals 25% and the probability of the realized threat (ARO) is once in every ten years, the calculation is expressed as ALE = ARO x SLE = ARO x AV x EF = 0.1 x (1,000 x 0.25) = 25. Hence, the mitigation cost per year should not exceed the ALE value of 25 units. However, this is a very rough estimation since it might be difficult to estimate the costs for potential damage. Still, it is not feasible if mitigation costs are higher than the monetary value of the asset that is being protected; and it should not exceed the financial loss for a realized threat.

The residual risk, i.e., the remaining risk after a countermeasure has been applied, must also be considered. It could be acceptable to decrease the mitigation cost by increasing the risk. However, risk management is usually determined by the mitigation, transference, avoidance, or acceptance of the risk. Transferring the risk to an insurance company is also an option. Avoiding risk can be done by terminating the activity, which introduces the risk or by accepting that the risk might be necessary if mitigation is not feasible. This phase produces a document that contains justified risk handling related to each vulnerability and threat in all six STRIDE lists.

2.3.4 Test Phase

This phase consists of practical security testing and verifying the output from the predict and mitigate phases (i.e., the filtered STRIDE list and mitigations therein). These tests could also reveal more vulnerability. When possible, the use of automated software and hardware tools is recommended. The following three tests are used: fuzz testing, vulnerability testing, and penetration testing.

Fuzz testing considers mostly negative testing, i.e., testing inputs that are unexpected. Positive testing (valid, expected output testing) is assumed to be a part of normal function testing and not necessarily part of security testing; however, security and normal function testing should be integrated. The difference between vulnerability testing and penetration testing is subtle, and in many cases they overlap. Vulnerability testing is defined more as an automated approach with scanning tools used to find vulnerabilities, whereas penetration testing is a manual approach wherein vulnerabilities that are difficult to automate (e.g., TOE-adapted attacks as tests) are tested. Penetration testing considers both black- and white-box testing. Also considered for tests are internal and external perspectives. An example of white-box testing against a vehicle's wireless local area network (WLAN) router as a TOE can be a vulnerability scan of the private internet protocol (IP) address (i.e., test access via the vehicle's internal Wi-Fi network). An example of black-box testing of the TOE might instead be a vulnerability scan of the official IP address (i.e., to test access from the Internet). An example of an operating system specialized in penetration testing is Kali Linux, which contains many useful tools [41]. An example of a hardware tool, applicable for evaluation of an access point, is Pineapple [42]. The output from this phase results in a document verifying mitigations toward vulnerabilities found in this phase and the six STRIDE lists.

2.4 Integration into the vehicle

At this point, the test phase has mainly considered the testing of a device (e.g., a single ECU). As previously mentioned, it is important to consider how the TOE affects, and is affected by, other assets (e.g., ECUs) in the vehicle. The TOE is not changed in this step; rather, the focus shifts toward a broadening of the scope to consider the complete vehicle relative to the TOE. Hence, when the test phase passes the device tests, the focus shifts and turns once more toward the integration tests. A compromised ECU (e.g., the vehicle infotainment system) might be used as a platform to send critical CAN messages to other safety-critical ECUs, if not correctly isolated or if the communication is not filtered. An example of tests that could be performed against the vehicle network as a whole (including the TOE) is an attack that exploits the error handling at the CAN bus by injecting messages, eventually forcing one or many ECUs to shut down [43]. A test such as this can be automated with scripting languages (e.g., Python or Communication Access Programming Language). Scripting and tools are typically adapted to the TOE; an example of an effective analysis tool for the total vehicle network is CANoe from Vector [44].

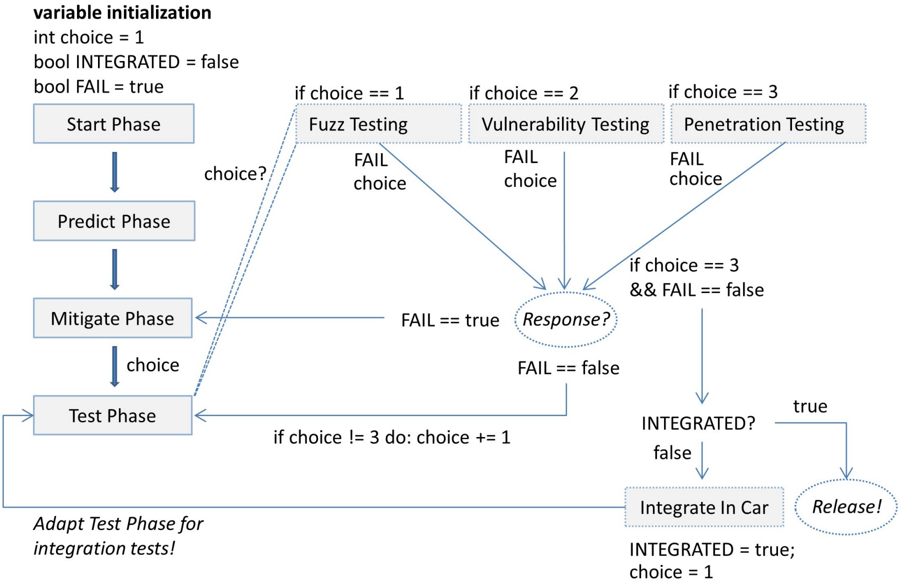


Figure 2.6: Flowchart of SPMT phases

2.5 Flowchart and Pseudocode for the whole process

In this section, all phases are visualized in a complete manner by combining stated diagrams, as shown in Figure 2.6, with the pseudocode shown in Figure 2.7. In the pseudocode, the usability of the SPMT methodology is displayed with the vehicle's WLAN router as a TOE; it is assumed that the start and predict phases have been completed. It is also assumed that the tools used in the test phase are Defensics, OpenVAS, and Kali Linux together with certain hardware tools. Hence, at row 1, the mitigate phase is entered. All identified vulnerabilities are corrected at row 2. The test phase is presented at row 3, with the variable choice set to fuzz testing (the first test to be performed). At rows 4 and 5, fuzz testing is entered. At row 6, the TOE is scanned with the Defensics tool. At rows 7-11, the response is handled. If the test fails, the mitigate phase is re-assessed, the vulnerabilities are addressed, and fuzz testing is performed again. If the fuzz testing is successful (row 11), the vulnerability testing (row 12) can begin. The vulnerability testing (rows 12-18) works in the same manner as the fuzz testing, except that the OpenVAS tool is used. Penetration testing, the last test, differs from the others because, if this test is successful, integration of the TOE to the car is performed (row 25); this can only be done once (row 24). At row 27, all tests are repeated once more (although they are adapted for integration testing) after the TOE is integrated into the vehicle. If all tests are successful, the product can be released to market (row 29). Within the industry, components are ordered by external suppliers in a more-or-less developed status. In these cases, it is possible to enter an appropriate phase, e.g., the last phase (test phase) to begin practical tests.

```
Mitigate Phase(choice):
  do:   mitigate vulnerabilities for choice;
  goto: Test Phase(choice);
Test Phase(choice):
    1:   //***Fuzz Testing***//
      do: scan TOE (use: Defensics)
         Response?
           CASE(FAIL):
              True ? goto: Mitigate Phase(1);
                 else
                 goto: Test Phase(2);
    2:   //***Vulnerability Testing***//
      do: scan TOE (use: OpenVAS)
         Response?
           CASE(FAIL):
              True ? goto: Mitigate Phase(2);
                 else
                 goto: Test Phase(3);
    3:   //***Penetration Testing***//
      do: attacks against TOE (use: Kali Linux
          hw Tools)
         Response?
           CASE(FAIL):
              True ? goto: Mitigate Phase(3);
                 else if !INTEGRATED
                    do: Integrate In Car and
                        INTEGRATED = true;
                    //***Repeat all tests***//
                    goto: Test Phase(1);
                       else
                       do: Release;
```

Figure 2.7: Pseudo code to exemplify the usability for the SPMT Methodology

Considering a released product, it is possible to assess SPMT as shown in Figure 2.8 by continuously monitoring for new threats or system changes.

For the former, new threats need to be assessed and placed into the appropriate list (predict phase), and, for the latter, system changes need to be reassessed with practical tests (test phase). Hence, the SPMT is adaptable and can meet the requirements to cover both development and after-market security analysis.

2.6 Discussion and Contributions

In this chapter, we highlight and discuss some of the advantages and benefits of the proposed methodology.

- *Innovative and important.* The SPMT methodology shows how components of existing models and methods of security can be applied successfully when securing connected cars. This approach is both innovative and important for the automotive industry.

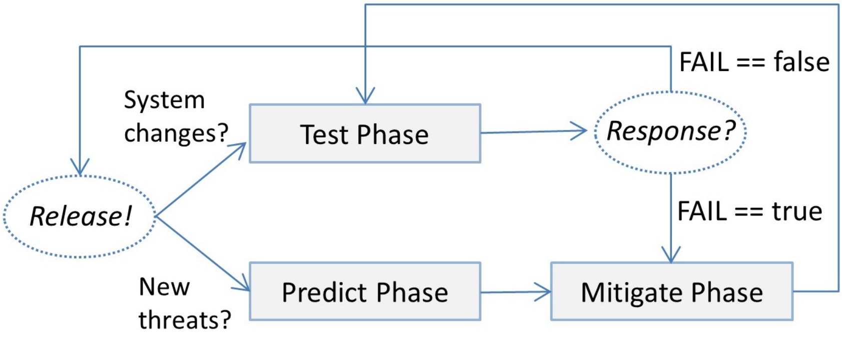


Figure 2.8: Possible scenarios after product is released to market

- *Comprehensive and systematic.* SPMT methodology offers a comprehensive systematic approach to security analysis specifically adapted for vehicles.

- *High coverage.* By following the SPMT methodology, comprehensive threat lists with mitigations are created for high-priority threats. High coverage is given for mitigations against threats with evaluated and prioritized risks considering the operational, safety, privacy, and financial factors that relate to vehicles. However, it still remains to be seen how large the coverage is in quantitative terms.

- *Security as entirety.* The SPMT methodology considers both the TOE from a device perspective and when the device has been integrated into the vehicle as a whole.

- *Cost and time effective.* The SPMT methodology simplifies the process of reassessing security after system changes and new threats. This is done by making use of existing documentation from earlier assessments and conducting practical tests concerning the changes in the test phase. Therefore, the SPMT methodology is both cost- and time-effective.

- *Adaptable.* The SPMT methodology is applicable for adaptation to different situations by selecting different tools and adapted attacks for different TOEs. It is possible to start the adaptation in different phases depending on the situation, as shown in Figure 2.8. Integrating the SPMT methodology to current development processes for the automotive industry is a straightforward approach, as exemplified in the Software V-model in Figure 2.9.

2.7 Conclusion

This article defines a security-enhancing, and thus safety-enhancing, methodology that identifies and mitigates vulnerabilities in vehicles. This is achieved through a comprehensive, systematic approach to security analysis, specifically adapted for vehicles. This methodology covers security analysis for the entire process, from product development to market introduction, by adapting and integrating relevant parts of existing security methods and models and by

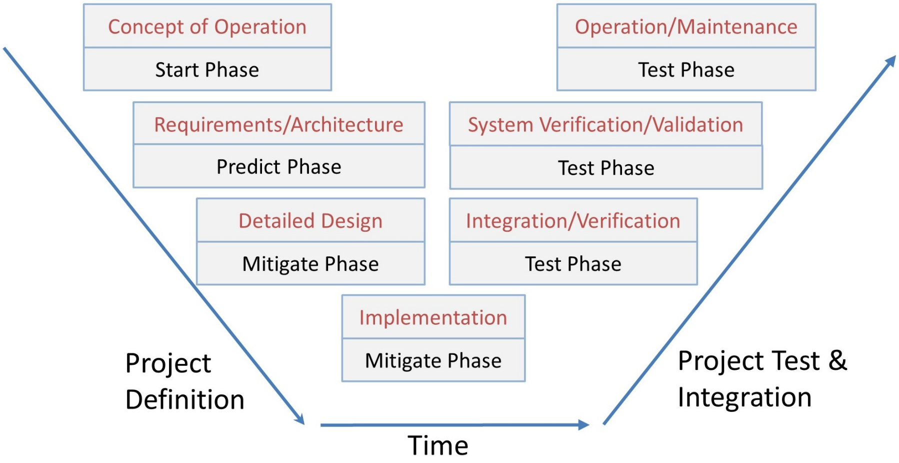


Figure 2.9: The SPMT methodology integrated into the software V-model process

incorporating new ideas suitable for vehicular domain. This methodology, named SPMT, is essential for the automotive industry's efforts to improve security and safety.

Chapter 3

REMIND: A Framework for the Resilient Design of Automotive Systems

Format-adapted version that appeared in IEEE Secure Development 2020

T. Rosenstatter, K. Strandberg, R. Jolak, R. Scandariato, T. Olovsson

Abstract. In the past years, great effort has been spent on enhancing the security and safety of vehicular systems. Current advances in information and communication technology have increased the complexity of these systems and lead to extended functionalities towards self-driving and more connectivity. Unfortunately, these advances open the door for diverse and newly emerging attacks that hamper the security and, thus, the safety of vehicular systems. In this paper, we contribute to supporting the design of resilient automotive systems. We review and analyze scientific literature on resilience techniques, fault tolerance, and dependability. As a result, we present the REMIND resilience framework providing techniques for attack detection, mitigation, recovery, and resilience endurance. Moreover, we provide guidelines on how the REMIND framework can be used against common security threats and attacks and further discuss the trade-offs when applying these guidelines.

3.1 Introduction

In the past years great effort has been spent in publishing guidelines and standards for security frameworks specific to their domains and in identifying security principles. Examples range from the NIST guideline for cybersecurity in smart grids [45], the cybersecurity guideline for ships [46], cybersecurity guidelines for the automotive domain [47–49] and the upcoming ISO/SAE standard for cybersecurity engineering for road vehicles, namely ISO 21434 [6].

Resilience is the next step towards reliable, dependable and secure vehicular systems. Vehicles need to be able to mitigate faults, errors, attacks and intrusions that would ultimately result in failures in order to withstand safety and security threats from their environment. We define automotive resilience as the *"property of a system with the ability to maintain its intended operation in a dependable and secure way, possibly with degraded functionality, in the presence of faults and attacks."* This definition is inspired by Laprie's definition [50] and the definition of network resilience by Sterbenz et al. [51]; however, the chosen definition highlights that faults or changes, e.g., functional and environmental (see [50]), can also be originated by an attacker whose aim is to disrupt the system.

Resilience can be obtained in many different ways and on different levels, i. e., hardware, software or (sub)-system level. Today's internal architecture of vehicles is quite complex and can be distributed over more than hundred so-called Electronic Control Units (ECUs). However, we are currently in a transition towards a more centralized architecture where functions will be concentrated on much fewer and more powerful ECUs [52]. These central ECUs are connected to sensors, actuators, external communication media and to some extent to smaller legacy subsystems. Such a centralized architecture enables vehicle OEMs not only to perform more resource intensive operations needed for autonomous driving, but also allows to introduce new designs and technologies needed to secure and protect these highly connected and autonomous vehicles. Virtualization is seen as one key technology enabling the isolation of vehicle functions from each other along with the possibility to dynamically assign hardware resources. Introducing resilience to such a centralized automotive system requires the deployment of techniques and principles in all layers and components of the system, ranging from the vehicle itself, the connected IT infrastructure, road infrastructure and the communication to other vehicles.

Motivation. The increasing complexity towards autonomous driving combined with the interconnectedness of vehicles, e. g., vehicle-to-vehicle and vehicle-to-infrastructure communication, and the continuous development of functions require vehicles to react and adapt to changes and attacks independently. The automotive domain is distinct from other domains as it is a safety and real-time critical system operated by millions of individuals each day. Furthermore, security and safety techniques need to be aligned and extended with resilience techniques in order to strengthen vehicles' capabilities to withstand impending threats.

Contributions. This paper provides a framework to design resilient automotive systems. First, we systematically identify relevant automotive resilience techniques proposed in the literature with the goal to provide a full picture of available tools and techniques. We also organize these techniques into a taxonomy, which comprises the categories of Detection, Mitigation, Recovery, and Endurance (REMIND). These categories represent high-level strategies that can help designers understand the *purpose* of each technique. Further, it can be beneficial to combine techniques from different strategies to achieve multiple layers of security. The selection of the right technique for the task at hand is further supported by associating the resilience techniques to the classes of *automotive assets* they are appropriate for. Additionally, we elaborate on the *trade-offs* (i.e., pros and cons) that are associated with each of the techniques, e.g., with respect to performance and other qualities. In summary, we provide a multi-dimensional decision support framework (built in a bottom-up fashion from the analysis of the literature) that can lead designers to the informed and optimal selection of a suitable set of resilience techniques to be implemented in an automotive system.

3.2 Methodology

By means of a systematic literature survey, we identify research papers that discuss techniques that are suitable to provide automotive resilience. We consider existing work related to resilience, fault tolerance and dependability. We also analyze the papers describing each technique to understand (i) the assets that can benefit from the technique, (ii) the risks that are mentioned as being mitigated by the techniques, and (iii) any pros/cons associated with the use of such technique.

We identified relevant research papers by searching the Scopus database [1]. A search string was intended to find relevant publications that carried out a review of suitable techniques. Therefore, we formulated the search string to *include* survey or literature review, and relevant topics, such as resilience, survivability, attack recovery, error handling or fault tolerance, as well as the keywords software, system or network. We *excluded* the keywords FPGA, memory, wireless, SDN and hardware to limit the search result to publications focusing on system architecture, software design or physical networks. Furthermore, we considered only publications written in English and published after 2010 in the areas of computer science and engineering. We manually screened the 200 most relevant publications returned by Scopus and found eight additional research publications, which were added to our result set. Ultimately, we retained and analyzed 12 publications which are shown in Table 3.1.

3.3 Attack Model and Assets

The four *strategies* in the REMIND framework are, as shown in Figure 3.1, further refined in *patterns* and *techniques*. A collection of these techniques specific for automotive systems is described in Section 3.4 and has been identified based on existing research in other domains and areas (see Table 3.1). We additionally describe the trade-offs of these techniques in Section 3.7 and point to relevant publications in Section 3.8. In the remainder of this section we describe the assets, security threats and attacks of automotive systems.

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

Table 3.1: Publications that provide an overview or collection of relevant techniques.

Discipline	Existing Work	Domain
Resilience	Chang2015 [53]	Fog Computing
	Hukerikar2017 [54]	High Performance Computing
	NIST 800-160v2 [55]	Systems Engineering
	Ratasich2019 [56]	Cyber-Physical Systems
	Sterbenz2010 [51, 57]	Networks
Security	Segovia2019 [58]	SCADA systems
Dependability	Bakhshi2019 [59]	Fog Computing
Fault Tolerance	Egwutuoha2013 [60]	High Performance Computing
	Kumari2018 [61]	Cloud Computing
	Mukwevho2018 [62]	Cloud Computing
	Slåtten2013 [63]	Software Engineering
	Wanner2012 [64]	Vehicle Controller

We consider four asset types, namely *Hardware*, *Software*, *Network/Communication* and *Data Storage.* The attacker aims to compromise these assets via various attack vectors, whereas the defender, i.e. the vehicle, aims to cope with these attacks via resilience techniques. We consider skilled attackers as well as novice hackers (e. g., script kiddies) and further give examples from an asset, threat and attacker perspective.

Hardware. Can be broken down to *ECUs*, *Sensors* and *Actuators.* An *ECU* can vary in complexity depending on its objective, from a specific limited task to a multitude of tasks. The former can relate to the processing of a sensor signal and the latter an infotainment-system with lots of applications. *Sensors* can give information about speed, temperature and obstacle distance and identification where the *Actuators* turn input from these sensors (via an ECU) into actions, such as braking, steering and engine control.

Attack example. Tampering with existing hardware or installing malicious hardware into the vehicle can act as mediators to gain complete vehicle control. Input signals from sensors may be manipulated to cause an unwanted behavior.

Software. Can be *in transit*, *at rest* or *running. In transit* can relate to software provisioning systems, such as over-the-air or workshop updates and the latter two to software installed or running in ECUs.

Attack example. Software vulnerabilities might be exploited, e. g., via a privilege escalation attack which enables ECUs to be re-programmed with additional functionalities, such as adding remote access to the system.

Network/Communication. Can be broken down to *internal* and *external communication.* Examples for internal communication are CAN, FlexRay, LIN, MOST and Automotive Ethernet and for external communication Wi-Fi, Bluetooth, and V2X as well as external interfaces such as OBD-II, debug ports (e. g., JTAG) and CD player.

Attack example. The attacker can try to inject malicious data, through a device connected to an in-vehicle bus affecting the internal communication. Furthermore, modification of V2X data from other vehicles as well as malicious roadside units (e. g., vehicle positioning or traffic condition data) could affect system functions.

Data Storage. Can potentially be sensitive data, such as cryptographic keys, forensics logs, system information (e. g., from software libraries, OS and

applications) and reports about the vehicle and the driver.
Attack example. The attacker can exploit secret keys used for sensitive diagnostics to disable firewalls. Logs and report data might be manipulated or removed to hide forensic evidence of the crime. Furthermore, information about the system can reveal vulnerabilities which might be exploited. Attackers typically exploit the above-mentioned assets in any order to achieve their goal, e. g., uploading malicious software to the vehicle by first compromising the cryptographic keys to get access to the memory and consequently upload a modified firmware containing malicious code. This can give elevated privileges and extended functionality which could cause inconsistencies or disruption of the system.

More examples of assets and related security threats and attacks can be found in Table 3.2.

Table 3.2: Automotive assets and related security threats and attacks

Asset	Asset Examples	Security Threat	Attack Examples
Hardware	ECU (hardware) Sensors Actuators	Disruption or direct intervention. Availability and Integrity.	*Fault Injection:* fuzzing, DoS, microprobing, malicious hardware as well as environmental injections (e. g., voltage and temperature) can disrupt or disable components or system resources. *Information Leakage:* side channel parameters, such as timing information or power consumption (e. g., differential power analysis) to extract secret keys.
Software	ECU (software) Libraries OS Virtualization	Manipulation of software, measurements or control signals. Availability and Integrity.	*Malware/Manipulated software:* indirectly affecting storage through alteration, deletion or blocking data, or indirect affecting the communication by read, manipulate or replay of messages, hence causing disruption and deviations from normal system operation.
Network/ Communication	CAN LIN MOST FlexRay Automotive Ethernet Mobile Network Wi-Fi Bluetooth OBD-II CD player	Communication failure or protocol vulnerabilities. Confidentiality, Integrity, Availability and Privacy.	*Fabrication/Jamming attack:* introducing fake traffic, e. g., sending high priority messages, to block legitimate low priority messages. *Masquerading/Spoofing attack:* masquerading as a legitimate node, e. g., by suspending the authentic ECU and send fabricated messages which seems to origin from the same. *Collision:* spoofing a message to induce a bit error/collision and then potentially spoof additional messages which get accepted. *Eavesdropping/hijacking:* intercept to read, block, manipulate or replay messages. *Suspension/DoS attack:* disable an ECU, such as inducing programming mode causing an ECU to not transmit or relay messages, potentially causing other ECUs to malfunction.
Data Storage	User Data Logs/Reports/Events Checkpoints Backups Forensics data Cryptographic material	Malicious handling of data storage. Confidentiality, Integrity, Availability and Privacy.	*Unauthorized read:* acquire sensitive data, such as privacy related user data e. g., previous locations or driving behavior. *Manipulation*: malicious alteration of data, e. g., replacing the software validation key enables potential alteration of memory data. *Removal:* data deletion of sensitive information, such as forensics data. *Reverse engineering:* extraction and analysis of firmware to deduce design features, vulnerabilities or secret keys.

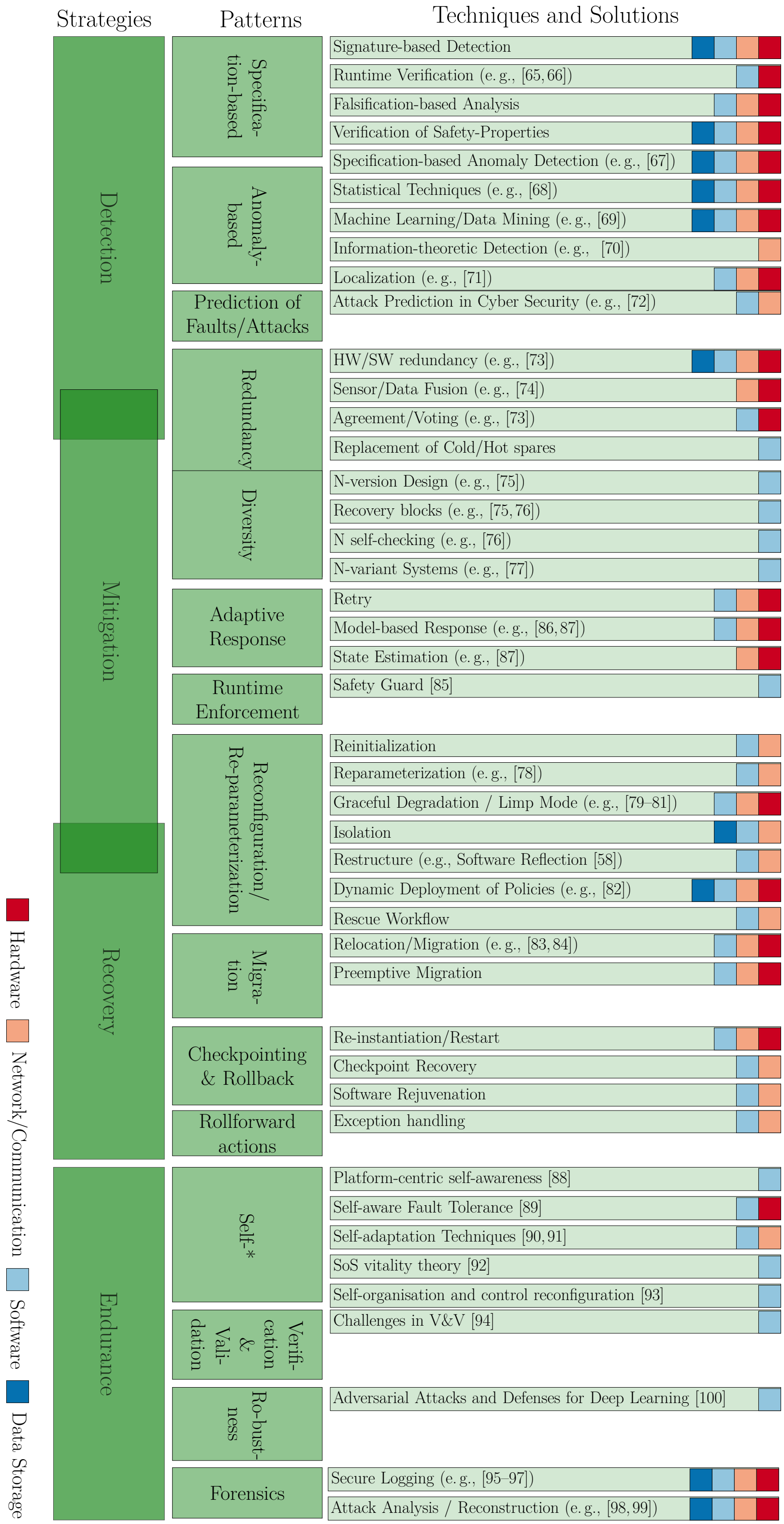
3.4 REMIND Automotive Resilience Framework

We have developed the REMIND framework shown in Figure 3.1 to provide system designers and developers with a categorization of suitable resilience mechanisms including the identification of the assets they protect. The structure of the layers is chosen similarly to the work in Hukerikar et al. [54], where the bottom layer is divided into *strategies* and the mid layer is split into *patterns* that provide more details about the way the strategies can be realized. We refer to relevant solutions for automotive systems in the top layer and further link to the survey papers and reviews that identify specific *techniques* for their domain in the description listed below.

The four REMIND strategies for providing resilience for vehicular systems are:

- **Detection.** Faults, attacks and other anomalies need to be detected by the system in order to take reactive measures to avoid a failure.
- **Mitigation.** Once an anomaly is detected and located, mitigation techniques need to be triggered to keep the system operational. These techniques may result in a non-optimal system state.
- **Recovery.** Transitioning back to the desired, i. e., optimum state, is the aim of recovery.
- **Endurance.** The focus is set on lasting resilience in contrast to recovery & mitigation strategies which aim at taking immediate measures.

The remaining part of this section details the strategies and describes the patterns and corresponding techniques.



Strategies	Patterns	Techniques and Solutions	Data Storage	Software	Network/Communication	Hardware
Detection	Specification-based	Signature-based Detection	✓	✓	✓	✓
		Runtime Verification (e. g., [65, 66])		✓		✓
		Falsification-based Analysis		✓	✓	✓
		Verification of Safety-Properties	✓	✓	✓	✓
		Specification-based Anomaly Detection (e. g., [67])	✓	✓	✓	✓
	Anomaly-based	Statistical Techniques (e. g., [68])	✓	✓	✓	✓
		Machine Learning/Data Mining (e. g., [69])	✓	✓	✓	✓
		Information-theoretic Detection (e. g., [70])			✓	
		Localization (e. g., [71])		✓	✓	✓
	Prediction of Faults/Attacks	Attack Prediction in Cyber Security (e. g., [72])		✓	✓	
Detection Mitigation	Redundancy	HW/SW redundancy (e. g., [73])	✓	✓	✓	✓
		Sensor/Data Fusion (e. g., [74])			✓	✓
		Agreement/Voting (e. g., [73])		✓		✓
		Replacement of Cold/Hot spares		✓		
Mitigation	Diversity	N-version Design (e. g., [75])		✓		
		Recovery blocks (e. g., [75, 76])		✓		
		N self-checking (e. g., [76])		✓		
		N-variant Systems (e. g., [77])		✓		
	Adaptive Response	Retry		✓	✓	✓
		Model-based Response (e. g., [86, 87])		✓	✓	✓
		State Estimation (e. g., [87])			✓	✓
	Runtime Enforcement	Safety Guard [85]		✓		
Mitigation Recovery	Reconfiguration/ Re-parameterization	Reinitialization		✓	✓	
		Reparameterization (e. g., [78])		✓	✓	
		Graceful Degradation / Limp Mode (e. g., [79–81])		✓	✓	✓
		Isolation	✓	✓	✓	
		Restructure (e.g., Software Reflection [58])		✓	✓	
		Dynamic Deployment of Policies (e. g., [82])	✓	✓	✓	✓
		Rescue Workflow		✓	✓	
Recovery	Migration	Relocation/Migration (e. g., [83, 84])		✓	✓	✓
		Preemptive Migration		✓	✓	✓
	Checkpointing & Rollback	Re-instantiation/Restart		✓	✓	✓
		Checkpoint Recovery		✓	✓	
		Software Rejuvenation		✓	✓	
	Rollforward actions	Exception handling		✓	✓	
Endurance	Self-*	Platform-centric self-awareness [88]		✓		
		Self-aware Fault Tolerance [89]		✓		✓
		Self-adaptation Techniques [90, 91]		✓	✓	
		SoS vitality theory [92]		✓		
		Self-organisation and control reconfiguration [93]		✓		
	Verification & Validation	Challenges in V&V [94]		✓		
	Robustness	Adversarial Attacks and Defenses for Deep Learning [100]		✓		
	Forensics	Secure Logging (e. g., [95–97])	✓	✓	✓	✓
		Attack Analysis / Reconstruction (e. g., [98, 99])	✓	✓	✓	✓

Figure 3.1: REMIND resilience techniques and solutions including a mapping to the assets for each technique. The overlap of the *Mitigation* strategy highlights that some patterns also contribute to *Detection* and *Recovery* respectively.

3.4.1 Detection

The monitoring and detection capabilities of a system can be limited due to various factors, such as computational resources, energy consumption, and the complexity of functions and network architecture. The move to a more centralized architecture, however, paves the way for more extensive monitoring.

Specification-based Detection. Malicious or abnormal behavior is detected using a specification that describes the behavior of signals or communication patterns. Domain knowledge is needed to create the specifications.

– *Signature-based Detection [56].* Signatures are constructed to describe known attack behavior. By design, these techniques suffer from detecting new attacks and zero-day vulnerabilities. However, they typically achieve a low false positive rate [67].

– *Runtime Verification [56, 63].* A monitor observes the system at runtime to verify the correctness of the execution. Formal specification languages, e. g., Signal Temporal Logic (STL) [101], have been developed to describe the normal system behavior which is matched against a trace during execution.

– *Falsification-based Analysis [56].* It extends STL by including a quantitative semantics allowing the return of real values rather than Boolean values.

– *Verification of Safety-Properties [56].* The formal verification of safety properties has become increasingly complex due to the added functionality in modern vehicles. Exhaustive verification techniques, as listed and argued by Ratasich et al. [56], are currently limited to small scale models.

– *Specification-based Anomaly Detection.* Normal behavior, according to a set of rules, is defined using this technique. An alert is sent when a violation of these rules is detected [67].

Anomaly-based Detection. Anomaly- or behavior-based detection techniques are based on comparing behavior with a model of normal behavior. Alerts are raised when a deviation is detected [102].

– *Statistical Techniques [56].* A statistical model describing the system or a specific process is designed in order to detect anomalies. Events are considered anomalies when the probability of their occurrence is below a certain threshold according to the model.

– *Machine Learning/Data Mining [56].* These techniques typically do not require domain knowledge. A model, such as Bayesian networks, neural networks and support vector machines, learns through training data how to classify observations in normal and abnormal classes.

– *Information-theoretic Detection [56].* The entropy of information can be used to detect anomalies, as a change of the entropy above a certain threshold may be caused by an attack, e. g., masquerading attack [56, 70].

– *Localization.* Finding the source of the attack may be required to take appropriate actions. Network-based Intrusion Detection Systems (IDSs) can be used to limit the location to a specific subnet, however, solutions identifying the particular ECU are needed (e. g., [71]).

Prediction of Faults and Attacks. First, the system needs to identify the presence of an attacker. The next actions are *attack projection* and *attack intention recognition* which aim at identifying the next steps and the ultimate goal of the attacker. *Attack or intrusion prediction* can be used to foresee when and where an attack will take place [72].

Adversaries mounting simpler attacks on a single vehicle, such as DoS attacks on the CAN bus, may be difficult to predict as the attack consists of fewer steps. However, large-scale attacks requiring the attacker to go through several stages may be predicted by this technique.

Redundancy. Redundancy is twofold, as it can support both detection and mitigation. It is important to highlight that purely redundant systems suffer from the same design faults and vulnerabilities. Thus, diversity is combined with redundancy to overcome this issue.

– *HW/SW Redundancy [54–56, 58, 60–63].* Redundancy combined with a voter allows to mask system failures. The voter compares the results of a number of independently executed software and/or hardware modules and selects, for instance, the majority [73]. Repeating the computation n times on the same hardware can be used to detect random faults.

– *Sensor/Data Fusion [56].* Data from different origins may be fused to compensate inaccuracies or temporary sensor failures. Sensor fusion, e. g., extended Kalman filter [103] and particle filter [74], can be used to describe the non-linear relationship between sensors. For example, the motion of a vehicle can be described with measurements from the wheel speed sensor, GPS location and data received from other vehicles.

– *Agreement/Voting [54, 56, 60, 63].* Redundant components are required for this technique. Voting can be realized in two ways, i. e., exact voting and inexact voting, where the latter allows a variation of the result within a certain range [73].

– *N-version Design [54–56, 60, 62].* N versions of a software with the same requirements are developed by N independent teams resulting in a diverse set of functionally equivalent software components that fulfill the same specification. These versions are executed concurrently and a voter decides based on the majority or calculates, for instance, the median or average of the results [75].

– *Recovery Blocks [54, 62, 63].* Similar to n-version design, n versions of a software component exist; however, only one version is executed at a time. After the active version is executed, a common acceptance test decides whether the result is accepted. In case the result is rejected, the subsequent version is executed and evaluated [75, 76].

– *N self-checking [60].* This technique is a combination of n-version design and recovery blocks. It requires at least two diverse versions with their own acceptance test. When the active component fails its acceptance test, the subsequent component takes over [76].

– *N-variant Systems.* Multi-variant execution automatically diversifies software and monitors the output of at least two variants to detect and mitigate attacks [77].

– *Replacement of Cold/Hot Spares [56].* Concurrent and sequential execution of redundant software components is costly in terms of energy consumption and computational resources. Therefore, the introduction of cold or hot spares, such as in N self-checking, have been found to be a viable alternative [56].

3.4.2 Mitigation

After detecting an attack or anomaly, the system needs to react to reduce the impact of the attack. Some mitigation techniques may require the transition to a non-optimal state.

Adaptive Response. We focus on techniques that adapt the response of a function or sub-system in order to maintain its intended functionality.

– *Retry [61, 62].* Performing the same computation with new measurements if the first computation resulted in an undesired system state or in an error. Retry can mitigate a replay attack.

– *Model-based Response and State Estimation [58, 64].* System models, e. g., Kalman filter for state estimation [103,104], or parameter estimation techniques, like regression analysis, are not only a temporary solution to mitigate attacks, such as replay and masquerading attacks, they can also be used to alert the system and log important information for forensics [87].

Runtime Enforcement. Runtime enforcement is an extension of runtime verification where the system also reacts to violations [65].

Reconfiguration and Reparameterization. The system protects itself by adapting parameters when an attack is detected. We distinguish between reconfiguration and migration in the way that migration focuses on relocating functionality whereas reconfiguration changes system or application parameters.

– *Reinitialization [54].* Temporary faults and attacks can be addressed with this technique. However, permanent faults or reoccurring attacks cannot be mitigated by restoring the system or a function to its initial state. Reinitialization can be seen as checkpoint recovery with the checkpoint being the initial state of the system or function.

– *Reparameterization [56].* Is similar to reinitialization, however, the system configuration is dynamically adjusted to the situation. As Ratasich et al. [56] point out, reparameterization typically results in a non-optimal state.

– *Graceful Degradation / Limp Mode [56, 58].* Given the extended automated driving functions of future vehicles, it is of utmost importance to implement more sophisticated solutions that ensure the passengers safety when key components in the vehicle fail or are subject to attacks. These techniques are similar to reparameterization, but focus on safety and should be seen as a last resort. Modern vehicles already have a so-called limp mode implemented, which is triggered when the vehicle detects major technical problems [105].

– *Isolation [54,56].* Restricting access or completely isolating system components in the presence of an error or intrusion can limit the impact on the entire system and its performance.

– *Restructure [54].* Restructuring components within a sub-system aims at providing resilience through reconfiguration of affected components. Segovia et al. [58] explore software reflection as means to mitigate attacks.

– *Dynamic Deployment of Policies [58].* Security or other policies can be applied dynamically based on the type of attack, e. g., DoS or masquerading, that is detected.

– *Rescue Workflow [61,62].* A workflow can be used to describe tasks with their dependencies to each other. The idea behind rescue workflows is to dynamically adjust the structure of the workflow when an error or intrusion affects a specific task. Existing cloud solutions may need to be adapted for automotive systems.

3.4.3 Recovery

Recovery techniques intend to bring the system back to an optimal state.

Migration. These techniques are mainly originating from high performance computing and cloud systems. As future automotive systems move towards a centralized architecture, virtualization and service-oriented architectures are becoming more relevant.

– *Relocation/Migration [56,62].* Virtualization such as hypervisor and container-based solutions allow a fast migration and relocalization to other nodes in the vehicular network.

– *Preemptive Migration [61, 62].* Continuous monitoring and analysis of the system can be used to relocate software functions or services before a fault occurs.

Checkpointing & Rollback. A checkpoint or snapshot describes the system state at a specific point in time. By design, recovery does not prevent the same attacks from happening again.

– *Re-instantiation/Restart [54, 56, 60, 62].* When an intrusion is detected, the affected component can be re-instantiated or restarted to recover to a known, error and attack free, state. This technique can be combined with reparameterization to avoid the same anomaly to happen again [56].

– *Checkpoint Recovery [54, 60–63].* Snapshots can be created in two ways: checkpoint-based and log-based. Egwutuoha et al. [60] highlight the complexity of taking checkpoints in a distributed system, as these checkpoints need to be consistent.

– *Software Rejuvenation [54, 62].* This technique carries out periodic restarts or reinitializations of the system to maintain a known, error-free state.

Rollforward actions. These techniques aim at bringing the system to a stable state immediately before the error or attack was detected. As in rollback, the recovery is based on using checkpoint-based or log-based recovery [54].

– *Exception Handling [54].* From a model-driven engineering view, *Rollforward* can be performed using exception handling. Slåtten et al. [63] highlight that this solution can be only applied to anticipated events.

3.4.4 Endurance

Resilience needs to be ensured over the entire lifetime of a vehicle. The preceding techniques center around providing immediate response when anomalies are detected.

Self-*. Self-* or self-X techniques cover solutions and research directions focusing on how to introduce autonomy into the system. This pattern is especially important for future vehicles as the environment is and will change frequently, new vulnerabilities will be found, new attempts to attack vehicles and their infrastructure will be developed, and new technologies will appear. Also, considering the lifetime of cars, which is around 10–15 years, it is evident that automotive systems need to adapt to a certain extent autonomously.

Verification and Validation. Due to the increasing functionality and interconnectedness of modern vehicles it is required to update software components via over-the-air updates in order to fix vulnerabilities and bugs or upgrade vehicle functions. This is especially challenging as each vehicle model can be further configured, resulting in a manifold of possible vehicle configurations.

Robustness. Artificial intelligence, especially machine learning, is a key technology for autonomous driving and decision making, as the system needs to be able to handle previously unseen situations [56].

Forensics. Providing evidence of intrusions even after a crash is important for taking appropriate countermeasures.

– *Secure Logging.* Hoppe et al. [95] express the need for forensic solutions in vehicles. Non-safety-critical events, such as updates, component failures and other malfunctions, need to be logged and stored securely for a prospective analysis. The authors also discuss in great detail which information and how this information can be stored in vehicles.

– *Attack Analysis.* Nilsson and Larson [98] specify requirements for forensic analyses of the in-vehicle network. It is also important to analyze attacks disclosed by researchers, such as Checkoway et al. [106] and Miller and Valasek [107], as well as attacks logged by the vehicle manufacturers in order to take appropriate actions.

3.5 Related Work

Making vehicles safe and secure has traditionally been the main focus in research. For instance, methods to combine safety and security [108] and how to assess an automotive system and/or derive security requirements and mechanisms have been proposed [37, 109, 110]. Le et al. [111] provide a survey on security and privacy in automotive systems and further provide an overview of suitable security mechanisms.

One of the first structured collections of principles for cyber resilience is the Cyber Resiliency Engineering Framework [112] by MITRE in 2011 which got further incorporated in NIST SP 800-160v2 [55]. Other work describing principles for resilience have been either concentrating on other domains, i. e., high performance computing, cyber-physical systems, or networks, or they focused particularly on dependability or fault tolerance. Table 3.1 provides an overview of relevant publications, which provide a comprehensive overview or collection of techniques, and categorizes them according to their discipline and the area they are focusing on.

The reviewed publications classify the identified techniques in different ways. Hukerikar et al. [54] divide them into strategies, i. e., fault treatment, recovery, and compensation, whereas Ratasich et al. [56] organize them according to their ability, i. e., detection and diagnosis, recovery or mitigation, and long-term dependability and security. Work focusing on fault tolerance either split the identified techniques in reactive and proactive measures [61, 62] or classify them according to their ability, e. g., error handling and recovery [60, 63].

With the developed REMIND framework, we contribute to supporting the resilience of automotive systems by: (i) identifying techniques for attack detection, mitigation, recovery, and resilience endurance; (ii) organizing the techniques into a taxonomy to guide designers when selecting resilience techniques; (iii) providing guidelines on how the REMIND framework can be used against common security threats and attacks; and (iv) discussing the trade-offs when applying the techniques that are highlighted in this framework.

In addition to the identified techniques in Figure 3.1, we point to implementations relevant for or specific to the automotive domain in Appendix 3.8.

3.6 Conclusion

The reviewed work shows the current research efforts towards making systems resilient to attacks and faults in related domains. We present a novel structure for categorizing resilience techniques in the form of the REMIND framework with the aim to lead designers in making informed decisions when choosing resilience techniques. We build upon the existing work and set the focus on the limitations of automotive systems and their challenges. The REMIND techniques have been chosen considering automotive assets and related attacks which are described in Section 3.3 and further linked to the guidelines and trade-off analysis in Section 3.7.

Future work includes the validation of the REMIND framework in regard to studying its applicability in industry in more depth. Furthermore, specific solutions for the identified techniques that consider the unique properties of automotive vehicles can be explored. Especially, the role of software-defined networking and its contribution to resilience can be investigated.

3.7 REMIND Resilience Guidelines

In this section, we report in Table 3.3 resilience techniques that can be used against common security threats and attacks. We also describe the trade-offs when implementing these techniques.

Table 3.3: REMIND Resilience Guidelines

Asset	**Attack**
Hardware	Fault Injection

Resilience Strategy/ Technique	**Trade-off**	
	Pros	Cons
DETECTION • Statistical Techniques [56] • Machine Learning/Data Mining [56] • Localization (e. g., [71]) • Sensor/Data Fusion [56]	• Less computation is required. • No domain knowledge is needed. It handles multivariate and non-linear data. • Identifies the exclusive part causing the fault or attack. • Calculates a value of trust of the data sources derived from the normalization factor.	• Very sensitive to outliers, imprecise detection, and increased complexity when modelling non-linear data. • Requires training. Imprecise prediction: false positives and false negatives. Time penalty and resource consumption (power, processing, and storage). • Often applied offline. The precision of the localization is dependent on both, the number of observed parameters and the set frequency for probing monitored resources. • Imprecise detection: false positives and negatives. It also introduces time penalty (increase in execution time) and space penalty (increase in resource usage).
MITIGATION • Hardware Redundancy [54–56, 58, 60–63]	• Enables offsetting the effects of faults and attacks, and allows the progress of the system without loss of functionality.	• Time penalty (increase in execution time) and resource consumption (increase in required resources). Hardware costs independent of whether attacks occur. Also, the design and verification of replicas requires an effort.
RECOVERY • Relocation/Migration [56, 62]	• Maintain system functionality in an operational state as it was before the fault or attack.	• May cause a degraded system, with less functionality, resources, and performance.

Table 3.3 – *Continued on next page*

Table 3.3 – *Continued from previous page*

Resilience Strategy/ Technique	**Trade-off**	
	Pros	Cons
ENDURANCE • Self-aware Fault Tolerance [89]	• Enables systems to adapt their behavior when a fault or attack occurs in their environment, thus allowing a continuous operation of these systems.	• Complexity and resource consumption.
Asset Software	**Attack** Malware/Manipulated Software	
DETECTION • Signature-based Detection [56] • Runtime Verification [56, 63]	• A precisely calibrated signature effectively identifies abnormal events during software execution. • Well-established and efficient technique to verify the correctness of software execution and monitor the behavior of the system.	• Does not work when designers and 3rd party suppliers (e.g., intellectual property providers) are not trusted. It cannot handle zero-day attacks and, thus, often used in combination with anomaly-based techniques leading to an increased resource consumption and time penalty. • Limited coverage. The used monitoring algorithms usually handle a single execution trace which limits the scope of the verification.

Table 3.3 – *Continued on next page*

Table 3.3 – *Continued from previous page*

Resilience Strategy/ Technique	**Trade-off**	
	Pros	Cons
MITIGATION • Software Redundancy [54–56, 58, 60–63] • N-Version Design [54–56, 60, 62] • Agreement/Voting [54, 56, 60, 63] • Recovery Blocks [54, 62, 63] • N self-checking [60]	• Helps to contain and exclude malicious behavior (i.e., reduces likelihood of harm). Enable restoration in case of disruption. Enhances the availability of critical capabilities. • Helps to mitigate the impact of failures when a risk is introduced to system design or configuration. • Typically combined with redundancy. Can be used to select, for instance, the average or median of the results provided by the redundant sources. • Uses different implementations of the same design specification to provide tolerance of design faults. • Provides mitigation by creating N versions of the same software, each with its own acceptance test. The version that passes its own acceptance test is selected through an acceptance voting system.	• Resource consumption. It demands the protection of redundant resources. It can degrade over time as configurations are updated or connectivity changes. It is often applied with diversity techniques which increases complexity and leads to scalability issues. • Requires much effort for designing, implementation, testing, and validation of the N independent versions. • Attackers may exploit the voting process in order to force the system to a degraded mode. • Requires extra verification and validation effort and, thus, more resource consumption. It might be difficult to create alternative software implementations without any correlation between the various versions. • Causes an increase in required resources and execution time.

Table 3.3 – *Continued on next page*

Table 3.3 – *Continued from previous page*

Resilience Strategy/ Technique	**Trade-off**	
	Pros	Cons
RECOVERY • Preemptive Migration [61, 62] • Checkpoint Recovery [54, 60–63] • Software Rejuvenation [54, 62]	• Prevents failures from impacting running parallel applications by enabling the migration of running software from one virtual machine to another in real time. • Helps the system to resume its operation in a state free of the effects of the fault or attack. Frequent checkpointing reduces the amount of lost work. • Helps avoiding the costs of failures from software degradation, as periodic (graceful) restarts of the software component allow the release and re-allocation of memory, thus, operation in a clean state.	• Lack of standardized metrics for measuring and evaluating the health and interfaces between system components. • Overhead in relation to the size and frequency of created checkpoints. Creating a checkpoint, for instance, requires interrupting the normal operation of a system to record the checkpoint. Moreover, it requires storage resources to store the checkpoint. The created checkpoints might potentially contain an error or intrusion that has not been detected yet. Globally consistent checkpoints are not trivial to obtain in a distributed system, due to e. g., variation of the local clock, parallel computation and possible different system states. • Requires shutting the software down and restarting it periodically which causes the software to be unavailable for the duration of the restart. It is often a slow process requiring an extra overhead.

Table 3.3 – *Continued on next page*

Table 3.3 – *Continued from previous page*

Resilience Strategy/ Technique	**Trade-off**	
	Pros	Cons
ENDURANCE • Platform-centric Self-aware-ness [88] • Secure Logging (e. g., [95–97])	• Enables systems to recognize their own state and to continuously adapt to change, evolution, system interference, environment dynamics, and uncertainty. It optimizes resilience, quality of service, and supports system dynamics and openness. It also helps to reduce uncertainties and identify inconsistencies. • Prevents modifying the logs by using e.g., chained hashes. It enables storing security-related events containing information about e.g., flash operations, external interactions, and power downtime. This information helps to reconstruct events, detect intrusions and identify problems.	• Automatically maintaining coherent specifications that capture and monitor security is a challenging task. Complexity, scalability, and difficulty in dealing with uncertainties and inaccuracies. The determination of relevant dependencies in a complex system is also challenging. • Resource consumption and time penalty. Moreover, missing authentication and lack of cryptographic means to ensure data integrity can limit the potential of the logging.

Table 3.3 – *Continued on next page*

Table 3.3 – *Continued from previous page*

Resilience Strategy/ Technique	**Trade-off**	
	Pros	Cons
Asset Network/Communication	**Attack** Fabrication/Jamming	
DETECTION • Specification-based Anomaly Detection (e. g., [67]) • Localization (e. g., [71]) • Verification of Safety-Proper-ties [56]	• Helps detecting anomalies in the system's behavior by reporting the specific deviation that has been observed. • Identifies the exclusive part causing the fault or attack. • Ensures that the system does not evolve in unsafe state starting from some initial conditions.	• Needs of resources for detection and processing of collected information (e.g., costly intelligent sensors). Domain knowledge is required to specify normal behavior. Specifications need to be adapted for each specific vehicle configuration otherwise risk of high false positives or negatives. • Requires additional resources. • It is limited to small scale systems.
MITIGATION • Isolation [54, 56] • Restructure [54]	• It provides a remedy to enable the system to continue its operation by offsetting the effect of the attack. Also, it prevents loss of functionality. • Helps to mitigate incorrectness in the interactions between the components or subsystems by excluding the affected part from interacting with the rest of the system, and maintaining system functionality.	• Introduces a time penalty and an increase in required resources (e.g., replica modules that are used to compensate for isolating the affected component of the system). • May cause an operation of the system in a degraded condition which influences its performance and incurs additional time overhead to the system.

Table 3.3 – *Continued on next page*

Table 3.3 – *Continued from previous page*

Resilience Strategy/ Technique	**Trade-off**	
	Pros	Cons
RECOVERY • Relocation/Migration [56, 62] • Re-instantiation/Restart [54, 56, 60, 62]	• Maintain system functionality in an operational state as it was before the fault or attack. • Helps to restore the system to its initial state when the impact of the attack can not be handled in another manner. It guarantees that the impact of the attack is completely removed.	• May cause a degradation in the operation of the system which influences the performance and functionality thereof. • Restoring the system to its initial state causes lost data, such as privacy related data (e. g., location, speed, driving behavior) and workshop data (e. g., vehicle health, engine data and emissions). The impact of the lost data depends on the type of data and the current need for it. In addition, the re-instantiation of safety-critical functions may require the vehicle to be in standstill.
ENDURANCE • Self-adaptation [90, 91]	• Ensures a secure, reliable, and predictable communication between system components and between the system and its environment. Supports and maintains an acceptable level of service despite the occurrence of faults and other factors that affect normal operations. Seamlessly adapts to different network loads and reacts to security threats and other disturbances in the environment.	• Complexity and resource consumption.

Table 3.3 – *Continued on next page*

Table 3.3 – *Continued from previous page*

Resilience Strategy/ Technique	**Trade-off**	
	Pros	Cons
Asset Network/Communication	**Attack** Masquerading/Spoofing/Collision	
DETECTION • Information-theoretic Detection [56] • Falsification-based Analysis [56]	• Helps to detect anomalies by analyzing available audit logs and records (e.g., entropy measures) and comparing these records with defined normal behaviors. More records enhance the precision of the detection. • Provides an indication (i.e., a robustness degree) to what extent temporal logic properties are from satisfying or violating a specification.	• Time penalty for processing audit records. More records at disposal increases the processing time and complexity. On the other hand, a low number of records leads to an imprecise detection with more false positives and false negatives. • Imprecise detection: false positives and false negatives
MITIGATION • Rescue Workflow [61, 62] (adaptation may be necessary) • Dynamic Deployment of Policies [58]	• Enables the system to continue operation after the failure of the task until it is unable to proceed without amending the fault or attack. Already finished tasks do not need re-execution, thus saving time and resources. • Takes the dynamic and changing nature of attacks into account. Deploys different defense policies depending on the attack, for example, it can modify the executed actions while the attack is going on.	• It may lead to a decrease in the quality of service. Time penalty might be caused by recomputing and migrating the tasks which cause the problem. • Leads to performance overhead. Moreover, it always requires runtime permissions which may not be present when running normally. Complexity.

Table 3.3 – *Continued on next page*

Table 3.3 – *Continued from previous page*

Resilience Strategy/ Technique	**Trade-off**	
	Pros	Cons
RECOVERY • Checkpoint Recovery [54, 60–63] ◦ Re-instantiation/Restart [54, 56,60,62]	• Helps the system to resume its operation in a state free of the effects of the fault or attack. Frequent checkpointing reduces the amount of lost work. ◦ Helps to restore the system to its initial state when the impact of the attack can not be handled in another manner. It guarantees that the impact of the attack is completely removed.	• Overhead in relation to the size and frequency of created checkpoints. Creating a checkpoint, for instance, requires interrupting the normal operation of a system to record the checkpoint. Moreover, it requires storage resources to store the checkpoint. The created checkpoints might potentially contain an error or intrusion that has not been detected yet. Globally consistent checkpoints are not trivial to obtain in a distributed system, due to e. g., variation of the local clock, parallel computation and possible different system states. ◦ Restoring the system to its initial state causes lost data, such as privacy related data (e. g., location, speed, driving behavior) and workshop data (e. g., vehicle health, engine data and emissions). The impact of the lost data depends on the type of data and the current need for it. In addition, the re-instantiation of safety-critical functions may require the vehicle to be in standstill.
ENDURANCE • Secure Logging (e. g., [95–97])	• Prevents modifying the logs by using e.g., chained hashes. It enables storing security-related events containing information about e.g., flash operations, external interactions, and power downtime. This information helps to reconstruct events, detect intrusions and identify problems.	• Resource consumption and time penalty. Moreover, it requires authentication and cryptographic means to ensure data integrity and confidentiality.

Table 3.3 – *Continued on next page*

Table 3.3 – *Continued from previous page*

Resilience Strategy/ Technique	**Trade-off**	
	Pros	Cons
Asset Network/Communication	**Attack** Hijacking/Replay/Suspension/DoS	
DETECTION • Signature-based Detection [56] • Verification of Safety-Proper-ties [56]	• A precisely calibrated signature effectively identifies abnormal events during software execution. • Ensures that the system does not evolve in unsafe state starting from some initial conditions.	• Does not work when designers and intellectual property providers are not trusted. It cannot handle zero-day attacks and, thus, often used with Anomaly-based techniques leading to a increased resource consumption and time penalty. • It is limited to small scale systems.
MITIGATION • Reparameterization [56] • Isolation [54, 56] • Graceful Degradation [56, 58]	• Enables adaptation by switching the configuration parameters of the compromised component to another configuration. • It provides a remedy to enable the system to continue its operation by offsetting the effect of the attack. Also, it prevents loss of functionality. • Prevents a catastrophic failure of the system. It enables a system to continue functioning even after parts of the system have been compromised. It shuts down less critical functions to allocate the resources to more critical functions to maintain availability.	• Decreases the quality of service. • Introduces a time penalty and an increase in required resources (e.g., replica modules that are used to compensate for isolating the affected component of the system). • Causes a degradation in the performance of the operations and services of the system.

Table 3.3 – *Continued on next page*

Table 3.3 – *Continued from previous page*

Resilience Strategy/ Technique	**Trade-off**	
	Pros	Cons
RECOVERY • Relocation/Migration [56, 62] • Software Rejuvenation [54, 62] • Reinitialization [54]	• Maintain system functionality in an operational state as it was before the fault or attack. • Helps avoiding the costs of failures from software degradation, as periodic (graceful) restarts of the software component allow the release and re-allocation of memory, thus, operation in a clean state. • Applied in conditions in which the mitigation is deemed impossible. Restores or pristine resets the system to its initial state.	• May cause an operation of the system in a degraded condition which influences its performance. • Requires shutting the software down and restarting it periodically which causes the software to be unavailable for the duration of the restart. It is often a slow process requiring an extra overhead. • Causes loss of work, and accordingly leads to a waste of resources.
ENDURANCE • Attack Analysis / Reconstruction (e. g., [98, 99])	• Helps to enhance resilience by systematically and empirically analyzing attacks as well as used technologies (potential entry point, e.g., Bluetooth and WiFi) that interact with the external environment.	• Resource consumption and analysis effort.

Table 3.3 – *Continued on next page*

Table 3.3 – *Continued from previous page*

Resilience Strategy/ Technique	**Trade-off**	
	Pros	Cons
Asset Data Storage	**Attack** Unauthorized Read/Manipulation	
DETECTION • Signature-based Detection [56] • Specification-based Anomaly Detection (e. g., [67])	• A precisely calibrated signature effectively identifies abnormal events during software execution. • Helps detecting anomalies in the system's behavior by reporting the specific deviation that has been observed.	• Does not work when designers and intellectual property providers are not trusted. It cannot handle zero-day attacks and, thus, often used with Anomaly-based techniques leading to a increased resource consumption and time penalty. • Needs of resources for detection and processing of collected information (e.g., costly intelligent sensors). Domain knowledge is required to specify normal behavior. Specifications need to be adapted for each specific vehicle configuration otherwise risk of high false positives or negatives.
MITIGATION • Redundancy [54–56, 58, 60–63] • Isolation [54, 56]	• It enables data backup and restore by replicating information and data sources. • It provides a remedy to enable the system to continue its operation by offsetting the effect of the attack. Also, it prevents loss of functionality.	• Requires extra resources for data storage. • Introduces a time penalty and an increase in required resources (e.g., replica modules that are used to compensate for isolating the affected component of the system).
RECOVERY • Dynamic Deployment of Policies [58]	• Takes the dynamic and changing nature of attacks into account. Deploys different defense policies depending on the attack, for example, it can modify the executed actions while the attack is going on.	• Leads to performance overhead. Requires runtime permissions which may not be present when running normally. Complexity.

Table 3.3 – *Continued on next page*

Table 3.3 – *Continued from previous page*

Resilience Strategy/ Technique	**Trade-off**	
	Pros	Cons
ENDURANCE • Secure Logging (e. g., [95]) • Attack Analysis / Reconstruction (e. g., [98, 99])	• Prevents modifying the logs by using e.g., chained hashes. It enables storing security-related events containing information about e.g., flash operations, external interactions, and power downtime. This information helps to reconstruct events, detect intrusions and identify problems. • Helps to enhance resilience by systematically and empirically analyzing attacks as well as used technologies (potential entry point, e.g., Bluetooth and WiFi) that interact with the external environment.	• Resource consumption and time penalty. Moreover, missing authentication and lack of cryptographic means to ensure data integrity can limit the potential of the logging. • resource consumption and analysis effort.

3.8 Proposed Automotive Solutions

In Table 3.4 we provide a description of the solutions referred to in Figure 3.1. This overview of specific solutions should be considered as a starting point for interested readers and is by no means complete.

Table 3.4: Techniques and solutions relevant for the automotive domain.

DETECTION

Pattern	**Technique**	**Solution**
Specification-based	Runtime Verification	Heffernan et al. [66] use the automotive functional safety standard ISO 26262 as a guide to derive logical formulae. They demonstrate the feasibility of their proposed runtime verification monitor with an automotive gearbox control system as use case.
	Specification-based Anomaly Detection	Müter et al. [67] describe eight detection sensors that are applicable for the internal network of automotive systems. Six of these sensors are specification-based, e. g., the frequency of specific message types and the range of transmitted values like speed.
Anomaly-Based	Statistical Techniques	Nowdehi et al. [68] propose an IDS that learns about the automotive system by learning from samples of normal traffic without requiring a model definition.

Table 3.4 – *Continued on next page*

Table 3.4 – *Continued from previous page*

Pattern	**Technique**	**Solution**
	Machine Learning	Hanselmann et al. [69] propose CANet an unsupervised IDS for the automotive CAN bus. The anomaly score is calculated using the error between the reconstructed signal and the true signal value.
	Information-theoretic	Müter et al. [70] design an entropy-based IDSs for automotive systems with experimental results using data from a vehicle's CAN-Body network.
	Localization	Cho and Shin [71] present a scheme identifying the attacking ECU based on fingerprinting the voltage measurements on the CAN bus for each ECU. We see great opportunities in the localization of attacks when considering a centralized vehicle architecture combined with virtualisation techniques. This allows us to get detailed performance metrics of virtualized vehicle functions.
Predicting Faults and Attacks	Attack Prediction	Husák et al [72] perform a survey about current attack projection and prediction techniques in cybersecurity.
Redundancy	Diversity Techniques	Baudry and Monperrus provide in their survey [113] an overview of different software diversity techniques.
	Adaptive Software Diversity	Höller et al. [78] introduce an adaptive dynamic software diversity method. The diversification control receives error information from the decision mechanism and randomizes specific parameters during execution. Their experimental use cases demonstrate the dynamic reconfiguration of ASLR parameters, respectively, random memory gaps.
MITIGATION		
Adaptive Response	Model-based Response	Cómbita et al. provide a survey on response and reconfiguration techniques for cyber-physical control systems. Controllers or other systems that can be modelled as a control loop can be, for instance, adjusted to have another module in the feedback loop that compares the actual feedback from the control loop with a simulated/modelled response of what is expected.
Runtime Enforcement	Safety Guard	Wu et al. [85] show how so-called safety guards can be applied to safety-critical Cyber-Physical Systems (CPSs).
Reconfiguration and Reparametrisation	Graceful Degradation	Dagan et al. [79] provide an architectural design on how to extend limp modes so that they can be additionally used in a cyber security context. A safe-mode manager sends out triggering messages that cause the ECUs to transition to a limp mode when cyber-breaches are detected.

Table 3.4 – *Continued on next page*

Table 3.4 – *Continued from previous page*

Pattern	**Technique**	**Solution**
		Ishigooka et al. [80] propose a graceful degradation design process for autonomous vehicles with focus on safety.
		Reschkka et al. [81] explore how skills and ability graphs can be used for modelling, on-line monitoring and supporting decision making of driving functions.
	Restructure	Segovia et al. [58] set the focus of their survey on software reflection as mitigation technique for SCADA systems. Software reflection enables the system itself to examine and change its execution behaviour at runtime, which allows, for instance, the system to take actions when an attack is detected. The drawbacks currently seen in software reflection are the performance overhead, the increased execution time and the extended permissions required by software reflection.
	Dynamic Deployment of Policies	Rubio-Hernan et al. [82] propose an architecture for CPS that combines feedback control loops with programmable networking in order to mitigate attacks by re-routing traffic or applying security rules.
RECOVERY		
Migration	Relocation/Migration	Jiang et al. [83] propose a hypervisor that meets real-time requirements.
		Other relocation techniques are microservices [114]. Pekka and Mattila [84] propose a service-oriented architecture for real-time CPSs.
	Pre-emptive Migration	Engelmann et al. [115] describe a pre-emptive migration technique which uses a feedback-loop for observing health parameters to detect behaviour indicating a fault. This solution was developed for high performance computing and its applicability for the automotive domain needs to be further investigated.
Checkpointing and Rollback	Software Rejuvenation	Romangnoli et al. [116] describe a method to decide when it is safe to reload the software of a CPS.

Table 3.4 – *Continued on next page*

Table 3.4 – *Continued from previous page*

Pattern	Technique	Solution
ENDURANCE		
Self-*	Continuous Change	Möstl et al. [88] identify in their work the challenges of continuous change and evolution of CPS and propose two frameworks for self-aware systems centring around self-modelling, self-configuration and self-monitoring. The controlling concurrent change (CCC) framework is concerned with how to deal with changes in software components during the lifetime of a CPS. The authors highlight that the well-established V-model currently used is not designed for continuous change and therefore parts of the integration testing and system validation and verification need to be moved to the system itself. The proposed framework includes an automated integration process for new or updated functions that addresses safety, security, availability and real-time requirements. The structure and workflow of the proposed framework is further described using an automotive use case. The second framework concentrates on optimising performance, power consumption and resilience of CPS by using self-organisation and self-awareness techniques.
Verificaton & Validation	Challenges in V&V	De Lemos [94] discuss research challenges of verification and validation for self-adaptive systems at runtime.
Robustness	Adversarial Attacks on DNN	Yuan et al. [100] give an overview of current adversarial attack and defence techniques for deep learning.
Forensics	Secure Logging	Lee et al. [96] describe T-Box a secure logging solution for automotive systems that makes use of the trusted execution environment in ARM TrustZone.
		Mansor et al. [97] propose a framework to log vehicle data, such as diagnostic transmission codes, via the mobile phone and store it on a secure cloud storage.
	Attack Analysis / Reconstruction	Nilsson and Larson [98] discuss the requirements for conducting forensic investigations on the in-vehicle network.
		Bortles et al. [99] present which types of data may be retained from current infotainment and telematic systems.

Chapter 4

Resilient Shield: Reinforcing the Resilience of Vehicles Against Security Threats

Format-adapted version that appeared in IEEE Vehicular Technology Conference 2021

K. Strandberg, T. Rosenstatter, R. Jolak, N. Nowdehi, T. Olovsson

Abstract. Vehicles have become complex computer systems with multiple communication interfaces. In the future, vehicles will have even more connections to e.g., infrastructure, pedestrian smartphones, cloud, road-side-units and the Internet. External and physical interfaces, as well as internal communication buses have shown to have potential to be exploited for attack purposes. As a consequence, there is an increase in regulations which demand compliance with vehicle cyber resilience requirements. However, there is currently no clear guidance on how to comply with these regulations from a technical perspective. To address this issue, we have performed a comprehensive threat and risk analysis based on published attacks against vehicles from the past 10 years, from which we further derive necessary security and resilience techniques. The work is done using the *SPMT* methodology where we identify vital vehicle assets, threat actors, their motivations and objectives, and develop a comprehensive *threat model.* Moreover, we develop a comprehensive *attack model* by analyzing the identified threats and attacks. These attacks are filtered and categorized based on attack type, probability, and consequence criteria. Additionally, we perform an exhaustive mapping between asset, attack, threat actor, threat category, and required mitigation mechanism for each attack, resulting in a presentation of a secure and resilient vehicle design. Ultimately, we present the *Resilient Shield* a novel and imperative framework to justify and ensure security and resilience within the automotive domain.

4.1 Introduction

The complexity of vehicles is increasing. Consequently, vulnerabilities which might be exploited increase as well. Attacks to vehicular systems can be realized: (i) *indirectly* via compromised devices e.g., phones, dongles, or workshop computers connected to vehicle interfaces; (ii) *directly* via physical interfaces e.g., debug ports and the OBD-II connector; and (iii) *remotely* via various malicious sources, such as rogue access points and compromised servers. It has been demonstrated that vehicle cyber-attacks e.g., physical attacks [23] and remote attacks [24] are potential threats that have to be taken seriously. As a case in point, Miller and Valasek [22] performed a successful remote attack on a Jeep Cherokee via the Internet taking control of its primary functions by exploiting an open port via a cellular channel, an attack that led to a recall of 1.4 million vehicles. In [117], researchers managed to get remote access to the CAN bus of a BMW by compromising its infotainment system, allowing them to execute arbitrary diagnostic requests. Vulnerabilities in phone applications paired to vehicles have been exploited by adversaries to track vehicles, unlock the doors and to start their ignitions [118–120].

Motivation. Securing a vehicle as an afterthought is cumbersome, considering both the complexity which constantly increases and the existing dependencies on current architectural design. Hence, it is imperative to consider security during the vehicle's complete life cycle from idea to cessation.

There are increased requirements towards ensuring a resilient vehicle design, in a way that a vehicle should be able to withstand various types of cyber-attacks, malfunctioning units, and other external disturbances. Consequently, the resilient design should be able to *prevent*, *detect*, and *respond* to cyber-attacks, something which is also in line with the UNECE regulation [121] and the upcoming standard for automotive cyber security ISO 21434 [122]. In short, *prevention* is accomplished with security controls, *detection* by identifying faults and attacks, and *response* are mechanisms related to handling the detected anomalies with the ability to restore and maintain operation. However, there is currently no clear guidance how to comply with the aforementioned regulations and standards from a technical perspective. The *start*, *predict*, *mitigate*, and *test* (*SPMT*) is a systematic approach for identification and mitigation of vulnerabilities in vehicles [29]. The aim of *SPMT* is to ultimately enhance the security of vehicles through their entire life cycle. In this paper, we use and extend the *SPMT* methodology to establish an in-depth resilient design model with imperative mitigation mechanisms.

Contributions. By applying the *SPMT* methodology, we performed a comprehensive threat and risk analysis of 52 published attacks against vehicles from the past 10 years. 37 of these attacks were considered significant due to their high risk and were thus further mitigated with imperative security and resilience techniques. In this process, we have developed a *threat model* for securing vehicles by identifying vital vehicle assets and the related potential threat actors, their motivations and objectives. Moreover, we have developed a comprehensive *attack model* created from the analysis of the identified threats and attacks, further filtered and categorized based on attack type and risk criteria related to the probability and consequences of the attack. We present a comprehensive summary of the result from applying the *SPMT* methodology, an exhaustive mapping between asset, attack, threat actor, threat category and resilience mechanism for each attack. Ultimately, we define necessary security and resilience enhancements for vehicles, the *Resilient Shield*, which also validates the effectiveness of the methodology. To the best of our knowledge, our result is both novel and imperative to justify and ensure security and resilience within the automotive domain.

4.2 Related Work

Good practices for security of smart cars [123], *Cyber security and Resilience of smart cars* [124], and *The Cyber security guidebook for cyber physical vehicle systems, SAE J3061* [125], provide guidelines regarding threat and risk assessment. *EVITA* [126] proposed a method for security, safety, and risk analysis of in-vehicle networks, whereas *HEAVENS* [127] proposed a security model based on security objectives from EVITA and security attributes from Microsoft *STRIDE* [128]. Rosenstatter et al. [129] continue with the result from an analysis such as HEAVENS and map the identified security demands to security mechanisms. However, this mapping focuses only on securing the in-vehicle network.

The *SPMT* methodology builds on existing methods, models and security principles that are applicable to different phases in a vehicle's life cycle. By adapting and incorporating relevant parts suitable for the vehicular domain, a comprehensive security and safety enhancement is achieved. Consequently, the *SPMT* methodology covers the vehicles entire life cycle, something which cannot be achieved with existing methodologies [29]. *SPMT* adopts Microsoft's *STRIDE* categorization [128] which enables a mapping of attacks to a category with associated security attributes. Thus, mitigation mechanisms can be considered for the attribute and consequently mitigate more than one attack. Additionally in *SPMT*, a reduction analysis is performed for critical threats by creating attack trees to connect the vulnerability with the threat, i.e., an attacker wanders from a leaf node (condition) to the root of the tree (attacker objective). Consequently, the closer to the root a countermeasure is placed, the more conditions are mitigated. Moreover, some conditions can be attained by more than one attack, hence a countermeasure can mitigate several attacks.

The REMIND framework [13] for vehicular systems provides a taxonomy for resilience techniques identified from a review of existing work. In this paper we take advantage of previous knowledge and new results by applying the *SPMT* methodology. In the next sections we present the detailed approach followed by the results.

4.3 Approach

We use the aforementioned *SPMT* model to perform a comprehensive threat modelling and risk assessment of published attacks to further map these threats and attacks to imperative security and resilience mechanisms.

The *SPMT* methodology has 4 phases: *Start*, *Predict*, *Mitigate* and *Test*. In this paper, we perform the first three phases on a Target Of Evaluation

(ToE) and analyze security threats and attacks as well as provide mechanisms for the mitigation thereof (see Figure 4.1).

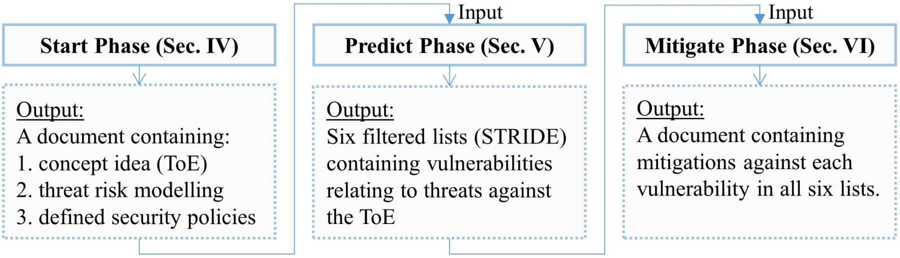

Input
Input
Start Phase (Sec. IV)
Predict Phase (Sec. V)
Mitigate Phase (Sec. VI)
Output:
A document containing:
1. concept idea (ToE)
2. threat risk modelling
3. defined security policies
Output:
Six filtered lists (STRIDE) containing vulnerabilities relating to threats against the ToE
Output:
A document containing mitigations against each vulnerability in all six lists.

Figure 4.1: The first three phases of the SPMT methodology

In the *Start Phase*, we address the following questions. *What are the threats requiring a resilient design? What are the entry points to the vehicle? Who are the actors, their motivators, and their objectives?* The outcome of the *Start Phase* is a threat model and high-level goals for the enforcement of security and safety attributes.

In the *Predict Phase*, we address the following question. *What are the potential attacks?* The outcome of the *Predict Phase* is an *attack model* which contains relevant attacks categorized and filtered according to a stated criteria.

In the *Mitigate Phase*, we address the following question. *What are the needed mechanisms to ensure a resilient design?* The outcome of the *Mitigate Phase* is a resilient design framework i.e., the *Resilient Shield*, which provides mechanisms and goals for detecting, preventing, and responding to security threats and attacks.

The *Test Phase* includes the implementation of the mitigation mechanisms followed by an execution of different security tests, such as fuzz, vulnerability, and penetration testing. In this paper, we do not perform the *Test Phase*; however, we plan to test the identified mitigation mechanisms within an industrial context in the future.

In the following sections, we perform and provide the outcomes of the first three phases of the *SPMT* methodology (see Figure 4.1) that are used to establish the *Resilient Shield*.

4.4 Threat Model

A threat model is created by considering: (i) the target of evaluation (ToE), and (ii) attackers as well as their motivators and objectives. First, our ToE is stated as the complete vehicle provided by the manufacturer, where we propose to include the following assets. As shown in Table 4.1, the relevance of these assets is verified by the mapping to attacks.

Internal and external communication: *Automotive Bus technologies*, e.g., CAN, FlexRay, LIN, MOST and Ethernet. *Connection interfaces*, e.g., OBD-II, USB, debug ports, Wi-Fi and Bluetooth.

Hardware: *ECUs*, e.g., sensor signal processing. *Sensors*, related to speed, position, temperature, airbag and object detection. *Actuators*, translate signals from ECUs into actions, e.g., braking, steering and engine control.

Software in transit, rest or running: *Software update systems*, e.g., over-the-air or workshop updates. *Software installed or running* in ECUs.

Data Storage: *Sensitive data*, e.g., cryptographic keys, forensics logs and reports.

Second, we propose a simplification of threat actors (i.e., attackers) inspired by the work of Karahasanovic et al. [130] in relation to motivators and objectives.

Actors and Motivators. *The Financial Actor* is driven by financial gain in relation to a company (intellectual property), organization or individual. This actor can be the owner who wants to make unauthorised modifications (e.g., chip tuning) or criminals who install ransomware. *The Foreign Country* is driven by power through cyber warfare, with the intent to disable viable assets within infrastructure (e.g., transportation). *The Cyber Terrorist* is driven by ideological, political or religious objectives. *The Insider* is motivated by retaliation or other personal gains, has knowledge of sensitive information and may plant malicious code into the vehicle. *The Hacktivist* is driven by publicity or adrenaline (i.e., the rush) and can have an agenda for political or social change. *The Script Kiddie* has usually no clear objective, possess limited knowledge and is often using already available tools and scripts. However, the reality is usually a combinations of the mentioned categories and objectives, and actors can be *black hat*, *gray hat*, or *white hat* hackers in relation to society's interpretations of the hackers' intentions. *White hat*, are assumed to be the good guys, *black hats* are the bad guys, and *grey hat* are somewhere in the middle.

Furthermore, in Section 4.6 we adopt the security and safety attributes used in *SPMT*. These attributes are imperative to uphold to ensure a secure and resilient vehicle. On the other hand, the actors are driven by stated *motivators* (e.g., financial, ideological, publicity) with the goal of compromising these attributes. A discussion and a brainstorming about fulfilment of these attributes is part of the *Start Phase*, however we have chosen to include it in Section 4.6 to have all considerations for mitigation in one section. Stated assets and actors are applied to Table 4.1 and used in the following section.

4.5 Attack Model

We perform a qualitative risk assessment of published attacks covered in news media and research publications by estimating (i) the probability and (ii) the consequences of the attacks based on the following criteria. As shown in Table 4.1, the affected assets, the threat actors and the STRIDE categories for each attack are considered during this assessment.

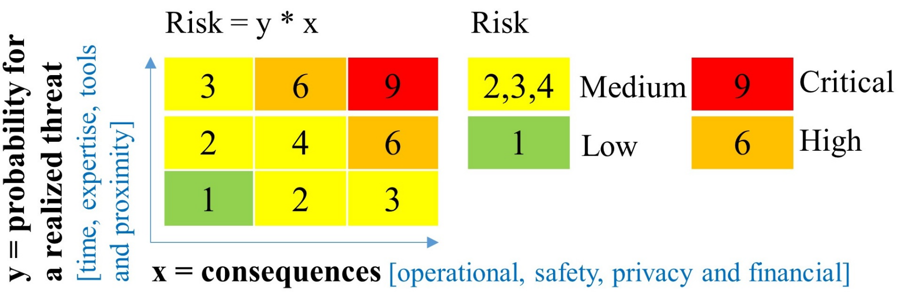
Attack Probability. The first step in this phase is to define attack probability where the three following estimates should be used:

E1: *Where, when, and in what situation can the attack be carried out?*
E2: *What expertise is required of the attacker?*
E3: *How much time does it take to perform the attack?*

The resulting probability is on a scale of 1 to 3, where 3 indicates that an attack is more probable to take place. The highest value in E1-E3 is chosen.

Attack Consequence. In the second step, the consequences are defined by assessing the effect of the attack on the operational, safety, privacy, and financial aspects. The resulting consequence is on a scale from 1 to 3, where 3 indicates that the consequence is more severe. The highest value is chosen.

Risk Assessment. Once we get the estimates of the attack probability and consequences, we estimate the overall risk by calculating the product of the probability and the consequence, which gives a risk value between 1 and 9 (see Figure 4.2). To achieve a realistic balance between the financial cost for mitigation and its related complexity versus the risk and asset value, we consider only the most significant threats. These threats have a risk value of 6 or 9, which is in line with ISO 26262 and ASIL [131] and corresponds to high and critical risk.



3	6	9
2	4	6
1	2	3

Risk = y * x
Risk
2,3,4
Medium
1
Low
9
Critical
6
High
y = probability for a realized threat
[time, expertise, tools and proximity]
x = consequences
[operational, safety, privacy and financial]

Figure 4.2: Adapted table for the risk calculation from the SPMT methodology.

4.5.1 Disclosed Attacks

To create the *attack model*, we follow the *SPMT* recommendation for search criteria and query scopus[1] and Google scholar for academic work, and common vulnerability databases (NVD, CVE) with keywords related to vehicle, attack and STRIDE categories (e.g., spoofing) or related terms (e.g., mitm). Moreover, we do query the Google search engine for media reports on attacks. Next, we classify the attacks according to STRIDE categories, followed by some examples. Attacks are considered and analyzed with respect to probability, consequence and risk within their respective category. Out of a total of 52 published attacks, we have identified 37 high and critical risk attacks which are further considered in this work.

1) Spoofing Attacks - Authenticity, Freshness [118,132–149]. The goal of the attacker is to intercept, hijack, manipulate or replay the communication with a potential remote access persistence. *Security flaws in mobile software*, such as demonstrated in the OwnStar attack [118]. OwnStar intercepts communication after the OnStar user opens the application, whereas the OwnStar device gains the user's credentials. Relay attacks, as in compromise of remote keyless entry systems as well as breaking poor authentication mechanisms [132–134]. GNSS spoofing considers broadcasting fake signals over authentic in order to to trick a receiver, with the intention to get a vehicle off course [135]. *In-vehicle protocol spoofing*, can affect safety critical actuators, such as brake, steering and engine control. Protocols themselves might lack inherent mechanisms for security

[1] `https://www.scopus.com/`

which makes active attacks possible such as malicious drop, modify, spoof, flood and replay of messages.

2) Tampering Attacks - Integrity [24,117,147,149–152]. Vulnerable USB/OBD-II dongles or compromised in-vehicle devices can potentially enable a hacker to control the communication. Devices can be compromised in various ways e.g., vulnerabilities in proprietary authentication mechanisms can enable the right to run sensitive diagnostics commands. Brute-force attacks can be used to retrieve cryptographic keys, with potential to upload exploits to ECUs. Physical tampering of ECUs or other connected devices. Manipulated firmware in current ECUs, such as malicious code injection via firmware update. Replacement of ECUs or new devices to eavesdrop/inject messages or to manipulate software, modify or compromise vehicle functions. Vulnerable connected devices such as OBD and USB dongles can potentially provide remote access to individual cars and vehicle fleets [151]. Moreover, in [24] firmware was extracted and reverse engineered, manipulated and injected directly into ECU firmware facilitating persistent and bridging capabilities for attacks.

3) Repudiation Attacks - Non-repudiation, Freshness. An attacker manipulates or removes forensic in-vehicle data, such as GPS coordinates, speed, acceleration and brake patterns, with the intention to hide traces of the attack. Despite our best effort, we did not find attacks which can be clearly mapped to this category; however, this type of attacks will likely be more frequent in the future due to both increased number of attacks and digital forensic investigations.

4) Information Disclosure Attacks - Confidentiality, Privacy [120, 149, 150, 153–156]. An attacker may be able to exploit cryptographic keys and consequently decrypt sensitive data by e.g., reverse engineering software with hard-coded keys. Bad routines for handling of replaced unit led to leaked sensitive data such as owners home and work address, calendar and call entries and Wi-Fi passwords [153]. A mobile application for vehicle control contained hard-coded credentials, thus an attacker may be able to retrieve sensitive data remotely by recovering the key from the application [120]. A vulnerability in an OBD-II dongle exposed all transferred data to the public [154]. Vulnerabilities in automotive bus technologies make various attacks possible, such as sniffing of CAN traffic due to its broadcast transmission and lack of encryption [155].

5) Denial of Service (DoS) Attacks - Availability [145–148, 157–160]. Many attacks focus on the in-vehicle network that uses CAN as this technology suffers from fundamental vulnerabilities with respect to security (e.g., broadcast communication, lack of encryption/authentication). Other attacks range from sending an indefinite amount of data to ECUs to make them unresponsive or crash, exploiting error handling mechanisms, or flooding the network with high priority messages in order to block lower priority messages. A vulnerability in the Bluetooth functionality supported unrestricted pairing without a PIN, thus enabled the potential for sending remote CAN commands affecting safety critical assets [159]. The Bus-off attack made ECUs unresponsive or crash [160]. Murvay et al. [158] managed to disable FlexRay nodes by exploitation of the bus guardian, power saving functionality and by causing loss of synchronization.

6) Elevation of Privilege Attacks - Authorization [22, 120, 147, 149, 150, 152, 161–163]. In [147] two Bluetooth vulnerabilities allowed remote code execution with root privileges. Moreover, manipulation of the firmware of the infotainment unit enabled injection of arbitrary CAN messages. In [161], they were able to release the airbag by message injection due to a vulnerable authentication mechanism. Lack of authentication in the NissanConnect app allowed to retrieve personal data by entering an URL with the vehicle identification number [163]. The outcome of this phase is applied to Table 4.1 and used in the next phase in the following section.

4.6 Resilient Shield

In this section we present the *Resilient Shield* which consists of high-level security goals emphasizing the overall design requirements resulting from an analysis of the threat model (Section 4.4). We further provide in Section 4.6.2 detailed directives for fulfilling the high-level security goals for resilient vehicles which are based on these goals and the *attack model* (Section 4.5). Table 4.1 summarizes the *Resilient Shield*. We list automotive assets, associate them with high risk attacks, potential threat actors and STRIDE threat categories, and link these to suitable security and resilience techniques to show how *Resilient Shield* can be used to mitigate these attacks.

4.6.1 High-level Security Goals (SGs)

The following high-level goals are the result of an analysis of the *threat model* detailed in Section 4.4. Each SG is associated with the relevant safety and security attributes they enforce.

SG.1 Secure Communication. *Integrity*, *authenticity* and, in specific cases, *confidentiality* need to be ensured for communication. *Integrity* and *authenticity* allow to verify the origin of the message and protect the message from being altered during transmission. *Confidentiality* can be achieved through encryption of the message to prevent unauthorized read access. *Freshness*, e.g., via counters or timestamps, can be used to mitigate replay attacks.

SG.2 Readiness. *Availability* to authorized entities under normal circumstances as well as disturbances. Even if an adversary tries to disrupt the information flow, the *integrity* and *availability* of correct information needs to be guaranteed.

SG.3 Separation of Duties is needed to limit access to resources for *authorized* entities only. *Authorization* should be combined with the principle of *least privilege* to limit the number of entities having access to a resource to the minimum.

SG.4 Secure Software Techniques need to provide security features to ensure that the executed software has not been modified by an unauthorized entity (*authenticity*) and that the software does not contain disclosed vulnerabilities.

SG.5 Separation/Segmentation on an architectural or process level is necessary in order to limit access and reduce the severity in case of an intrusion (*availability*). *Isolation* techniques, e.g., process isolation, should be considered where possible.

SG.6 Attack Detection and Mitigation is of utmost importance to enable the system to react and ideally prevent further damage to the system.

SG.7 State Awareness should be ensured with the ability to switch between various operational states, thus providing *reliability* and *maintainability*.

SG.8 Forensics is necessary for post analysis of detected malicious events and accordingly updating access control policies and other preventive measures. Physical security, such as vehicle locks, alarm system, and protecting infrastructure server rooms should be considered. Components must be extensively tested against requirements separately and when integrated into the vehicle, such as stated in the *SPMT Test Phase*. *SPMT* suggests to use both a qualitative and quantitative assessment; however, we focus on the qualitative assessment as the aim of *Resilient Shield* is to guide the resilient design of automotive systems. Moreover, a reduction analysis of attack trees is suggested to find commonalities in countermeasures; however this is not considered and is thus left as future work.

4.6.2 Detailed Directives

In this section, we list detailed techniques and patterns that contribute to the security and resilience of automotive systems based on the identified security goals, *threat* and *attack model* presented in this paper. First, we incorporate the identified patterns from the REMIND framework [13] in *Resilient Shield* and further extend them with security techniques to provide a comprehensive collection of both, security and resilience techniques for automotive systems. Second, we further discuss the security aspects of the identified resilience techniques. Next, we detail these techniques.

Authentication: Message authentication can be achieved through Message Authentication Codes (MACs) or signatures which ensure that the message: (i) is created by the claimed source and (ii) has not been altered during transmission. The authentication of devices can verify that the hardware, e.g., the head unit or a diagnostic device, is legit.

Encryption: Encryption of data ensures the protection of intellectual property, makes it more difficult to reverse engineer software, protects cryptographic material and the privacy of users and forensics data.

Redundancy/Diversity: A voting mechanism is used when comparing the output of two or more redundant systems or software functions. Redundancy increases the resilience against anomalies; however, from a security perspective it must be ensured that the voting process cannot be exploited by an attacker to perform DoS or spoofing attacks.

Access Control: Gateways with firewall capabilities allow filtering of messages between different networks in the vehicle. In addition, host-based firewalls on the ECUs can limit the exposure of open communication ports. Securing physical debug ports is vital to protect against unauthorized exploitation. Access control to resources such as files, computation, and diagnostic commands can be provided by the operating system or by e.g., challenge-response authentication.

Runtime Enforcement: Runtime verification is combined with reactive measures when safety properties are violated [13, 85].

Secure Storage: Cryptographic material needs to be protected against unauthorized modifications and read access. Data can be either stored encrypted in the regular file system or in a protected memory partition.

Secure Boot: A validation of the authenticity and integrity of the firmware to be loaded during the boot process [164].

Secure Programming: Secure programming guidelines such as MISRA

C [165] are important to avoid common programming errors. Additionally, trusted execution environments may be necessary for isolating and securing applications.

Secure Software Update: The ability to update software is not only a necessity to improve and extend functionality, it is also essential for security, e.g., to mitigate vulnerabilities. In addition, the update process itself needs to be secure [166], during the distribution and installation process.

Verification & Validation: The *Test Phase* in *SPMT* focuses on the need for security testing and verification of each asset by doing fuzz, vulnerability and penetration testing. In addition to security testing, the verification and validation of functionality and safety is required [13, 29].

Separation: Architectural separation can be achieved through physical separation into smaller networks or through virtualization techniques allowing to allocate resources to specific functions or systems.

Specification-based Detection: Knowledge about abnormal behavior is used to detect anomalies and attempts to exploit known vulnerabilities. It also requires domain knowledge and needs to be updated regularly [13, 167].

Anomaly-based Detection: Is based on defining normal behavior and deviations trigger alerts and has the potential to detect unknown attacks. Anomaly-based detection can be categorized in statistical, information-theoretic, machine learning and localization techniques [13, 167].

Prediction of Faults/Attacks: Predicting the next step or the ultimate goal of an ongoing attack.

Adaptive Response: The function response may be temporarily adapted, e.g., through a model, while under attack [13].

Reconfiguration: Graceful degradation can be used to limit the impact of an attack when preventive measures failed.

Migration: The ability to migrate services to other nodes in order to maintain system functions when under attack [13].

Checkpoint & Rollback: Used to recover the system to a desired state. The state needs to be secured, e.g., through secure logging, to defend against possible attacks that aim at modifying a saved system state [13].

Rollforward Actions: Upon detecting an anomaly or error the system transitions back to the state immediately before the event happened. Similarly to rollback it needs to be ensured that this mechanism cannot be exploited [13].

Self-X: The system needs to be aware of its state and able to switch to other states when anomalies occur [13, 168].

Robustness: Employed mechanisms and functions need to be robust against anomalies [13].

Forensics: Secure logging is used to record events, e.g., detection of an ongoing attack, use of specific services or diagnostics. In addition, events with non-repudiation claims can be used as evidence of a crime.

Table 4.1 presents the *Resilient Shield.* Assets with high or critical risk threats are associated with appropriate security and resilience techniques demonstrating the ability of *Resilient Shield* to defend against these threats. For example, hacktivists and insiders are the main threat actors for *communication:external:debugport*, such as JTAG, and needs to be protected with authentication mechanisms combined with access control or, if not possible otherwise, with physical protection (e.g., deactivation).

Table 4.1: Resilient Shield. A mapping from automotive assets to identified attacks, potential threat actors, STRIDE threat categories and ultimately to appropriate security and resilience techniques, and Security Goals (SGs).

Isolation Availability Authorization Confidentiality Maintainability Authenticity Freshness Privacy Integrity Reliability **Assets targeted by attacks with high or critical risk.** *ToE category:subcategory reference*	🟩 Resilience patterns identified in REMIND [13] **Potential Threat Actors** *Financial Actor (FA)* *Foreign Country (FC)* *Cyber Terrorist (CT)* *Insider (IN)* *Hacktivist (HA)* *Script Kiddie (SK)*	**STRIDE categories** *(S)poofing* *(T)ampering* *(R)epudiation* *(I)nformation Disclosure* *(D)enial of service* *(E)levation of privilege*	(SG.1,8) Authentication	(SG.1) Encryption	(SG.2) Redundancy/Diversity 🟩	(SG.3) Access Control	(SG.3) Runtime Enforcement 🟩	(SG.4,8) Secure Storage	(SG.4) Secure Boot	(SG.4) Secure Programming	(SG.4) Secure Software Update	(SG.4) Verification & Validation 🟩	(SG.5) Separation 🟩	(SG.6) Specification-based Detection 🟩	(SG.6) Anomaly-based Detection 🟩	(SG.6) Prediction of Faults/Attacks 🟩	(SG.6) Adaptive Response 🟩	(SG.6) Reconfiguration 🟩	(SG.6) Migration 🟩	(SG.6) Checkpoint & Rollback 🟩	(SG.6) Rollforward actions 🟩	(SG.7) Self-X 🟩	(SG.7) Robustness 🟩	(SG.8) Forensics 🟩
Hardware																								
sensor:camera [145, 146]	FC, CT, HA	S, D			●												●	●				●	●	
sensor:GNSS [135, 137, 140, 141, 143]	FC, CT, HA	S	●		●										●		●	●				●	●	
sensor:lidar [139, 145]	FC, CT, HA	S, D			●												●	●				●	●	
sensor:ultrasonic [146]	FC, CT, HA	S, D			●												●	●				●	●	
Communication																								
internal:can [151, 155, 157, 158, 160]	FA, FC, CT, IN, HA	S, T, I, D	●	●	●	●	●						●	●	●	●		●	●			●		●
internal:flexray [148]	FA, FC, CT, HA	S, D	●			●	●						●	●	●			●				●		●
external:bluetooth [117, 147]	FC, CT, HA	S, T, D, E	●			●							●											
external:usb [117]	FC, CT, HA	S, T, E	●			●							●											
external:keyfob [133, 134]	HA, SK	S	●			●									●								●	
external:wifi [118, 144]	HA, SK	S, I	●	●		●				●			●											
external:cellular [22, 117, 152, 156, 162, 163]	FC, CT, HA, SK	S, T, I, D, E	●			●							●											
external:obdII [120, 138, 142, 149, 151, 154, 157, 159]	CT, HA	S, T, I, D, E	●			●	●						●	●	●			●		●		●		●
external:debugport [22, 152]	HA, IN	I, E	●			●																		
Software																								
running:state [136]	FC, CT, HA	S, D				●				●				●	●					●	●			●
running:firmware [22, 117, 118, 144, 147, 150, 152, 156, 162, 163]	FC, CT, HA	S, T, E				●				●	●	●	●	●	●			●				●		●
instorage:update [117, 147, 152]	HA, SK	S, T, E	●	●		●		●	●		●		●	●	●	●	●	●	●					●
instorage:weakcrypto [132, 161, 163]	FC, CT, HA, SK	S, E	●							●													●	
Data Storage																								
crypto:certificates [152]	FC, CT, HA	I		●		●		●	●															
hw:replaced [153]	HA, SK	I	●	●		●																		

4.7 Conclusion

We have performed a comprehensive threat and risk analysis of published attacks against vehicles and derived imperative security and resilience mechanisms by applying the *SPMT* methodology. A *threat model* with vital vehicle assets and related potential threat actors, their motivations and objectives was developed. By an extensive analysis of threats and attacks, further filtered and categorized based on attack type, probability and consequence criteria, an *attack model* was developed based on the remaining high risk attacks. Based on the developed models, a comprehensive mapping between asset, attack, threat actor, threat category, and defense mechanisms was performed for all attacks and is presented in Table 4.1. Table 4.1 summarizes the outcomes by applying *SPMT*, i.e. the *Resilient Shield*, a novel framework both justifying and defining imperative security and resilient mechanisms needed in a modern vehicle. Consequently, the *Resilient Shield* can be used as a vital baseline for protection against common security threats and attacks.

We believe our work is imperative for facilitating and guiding the design of resilient automotive systems; however, it still remains to be seen how large the coverage is in relation to future attacks. Moreover, testing and validation of the *Resilient Shield* within an industrial context is left as a future work.

Chapter 5

A Systematic Literature Review on Automotive Digital Forensics: Challenges, Technical Solutions and Data Collection

Format-adapted version that appeared in IEEE Transactions on Intelligent Vehicles 2023

K. Strandberg, N. Nowdehi, T. Olovsson

A modern vehicle has a complex internal architecture and is wirelessly connected to the Internet, other vehicles, and the infrastructure. The risk of cyber attacks and other criminal incidents along with recent road accidents caused by autonomous vehicles calls for more research on automotive digital forensics. Failures in automated driving functions can be caused by hardware and software failures and cyber security issues. Thus, it is imperative to be able to determine and investigate the cause of these failures, something which requires trustable data. However, automotive digital forensics is a relatively new field for the automotive where most existing self-monitoring and diagnostic systems in vehicles only monitor safety-related events. To the best of our knowledge, our work is the first systematic literature review on the current research within this field. We identify and assess over 300 papers published between 2006 - 2021 and further map the relevant papers to different categories based on identified focus areas to give a comprehensive overview of the forensics field and the related research activities. Moreover, we identify forensically relevant data from the literature, link the data to categories, and further map them

to required security properties and potential stakeholders. Our categorization makes it easy for practitioners and researchers to quickly find relevant work within a particular sub-field of digital forensics. We believe our contributions can guide digital forensic investigations in automotive and similar areas, such as cyber-physical systems and smart cities, facilitate further research, and serve as a guideline for engineers implementing forensics mechanisms.

5.1 Introduction

The complexity of vehicles is increasing at a high pace. A modern vehicle can contain more than 150 ECUs (Electronic Control Units) and over 100M lines of code. Moreover, current vehicles have various connection interfaces and a large amount of forensically interesting data exchange between a multitude of entities such as sensors, actuators, ECUs, the Internet, and infrastructure. To enable a proper forensic investigation, data must be collected, stored, and processed in a forensically sound and secure manner.

In the remainder of this section, we explain the complexity of the vehicle architecture, its relationship to other similar areas, and the field of automotive digital forensics. We define the goal with our paper, problem, approach, and our main contributions.

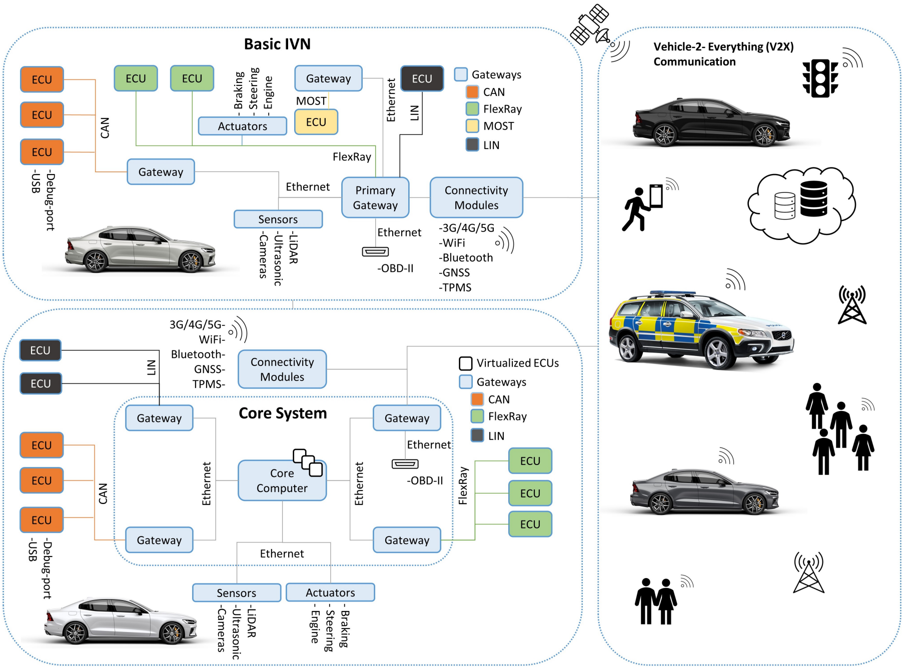


Figure 5.1: The In-Vehicle Network and V2X Communication

5.1.1 The Interconnected Vehicle

A vehicle uses various *hardware*, *software* and *storage* components, and different *communication* technologies.

Hardware can be broken down into ECUs, sensors, and actuators. The complexity of an ECU varies depending on its task, which can range from simple processing of sensor signals to an infotainment system with a multitude of applications. *Sensors* can give information about speed, temperature, distance, and identification of obstacles (e.g., pedestrians and animals). The *actuators* turn input from these *sensors* (via an ECU) into actions, such as braking, steering, and engine control. *Software* can be either *in transit*, *at rest*, or *running*, e.g., software provisioning systems, such as over-the-air or workshop updates, *transmits* software, and software can be installed (*at rest*) or be *running* in ECUs. *Data storage* includes storage of, e.g., cryptographic keys, forensics logs, system information, and reports about the vehicle and the driver.

Figure 5.1 shows two examples of In-Vehicle Networks (IVNs) with various nodes belonging to different bus technologies, where the basic topology in the top currently is most common. Transmission can occur over, e.g., CAN, FlexRay, MOST, LIN and Ethernet. A primary gateway connects to sub-gateways responsible for translating and relaying traffic to the correct network segment. The vehicle is connected to the outside world via various connection interfaces giving rise to Vehicle-to-Everything (V2X) communication. Wireless connections occur via, e.g., 3G/4G/5G, WiFi, and Bluetooth, and physical connections via, e.g., OBD-II, USB, and debug-ports. The communication is extensive considering the amount of data generated in the vehicle and the increasing communication with the outside world, such as with other vehicles, roadside units (RSUs), and with cloud-based services.

The other example in the lower part of Figure 5.1, shows that there is a trend in the automotive industry to move towards a more centralized architecture with fewer yet more powerful ECUs due to increased requirements for system performance in self-driving vehicles, where high-performance computers can virtualize hardware [169]. Automotive Ethernet communication becomes more prevalent due to its higher bandwidth and more straightforward adaptation to security features (e.g., authentication and encryption of messages) compared to other automotive communication technologies. Thus, the potential for better use of technical solutions for automotive forensics come to light.

ECUs have different functionalities and responsibilities, e.g., braking, steering, and engine control are part of the safety critical system. Advanced driver assistance systems (ADAS) technology handles automatic emergency braking, lane assistance, and driver monitoring.

5.1.2 Related areas

Internet of Things (IoT) relates to the automotive domain because of the increasing use of IoT devices in vehicles [170]. IoT refers to all devices with Internet capabilities such as cellular phones, medical devices, home appliances (e.g., TVs, washing machines, printers, etc.), including sensors and actuators. The concept of the Internet of Vehicles (IoV) strives to connect Vehicular Adhoc Networks (VANETs) to the IoT domain [171]. However, vehicles are safety-critical systems, and failures in any part of a safety-critical system have the risk of severe lethal outcomes for, e.g., drivers, passengers, and everyone around it.

For comparison, both IoT and the automotive domain suffer from previous designs not developed with security and forensics in mind. Instead, emphasis has been on cost and usability. However, safety has been imperative within automotive, but related security properties are often deficient. Moreover, there are no standardized processes for the forensic process within IoT or automotive. In IoT, data can be found on devices, networks, or the cloud [172]. The same applies to automotive. However, within automotive forensics, the main emphasis is on vehicles as safety-critical systems and the extensive communication during operation (cf. Table 5.3). Cooperative Intelligent Transport Systems (C-ITS) development has facilitated V2X communication and prepared to integrate vehicles into an upcoming smart infrastructure. ITS can refer to services for all modes of transport but relates mainly to transport on roads [173]. Future vehicles will become more autonomous and part of a much larger system with various interactions between a virtual and physical world. Services within ITS allow vehicles to share information, such as traffic conditions. Vehicles braking, speed, and position can be communicated, travel routes optimized, and traffic congestion limited. Virtual entities can replace physical entities, e.g., physical traffic lights and road signs can instead communicate traffic information wirelessly. Although there are similarities and interactions with other areas, the automotive domain is still very distinct and has its specific challenges. From an automotive digital forensic perspective, data communicated within the ITS is imperative to securely log and store to enable post-incident investigations.

5.1.3 Automotive Digital Forensics

Digital forensic investigations include the *identification*, *preservation*, *acquisition*, *verification*, *analysis*, and *reporting* of data [174]. The definition of digital forensics has expanded from involving just computers to include all digital devices that can store, process, or transmit data [175], a shift that leads to increased complexity. For example, when it comes to vehicles, file formats and operating systems differ in ECUs making it challenging to create unified standards and tools; and vehicle IVNs consist of many interconnected devices communicating using different communication protocols. Figure 5.2 shows some examples of potential entry points that can be of interest to cyber criminals.

It is well known that increased complexity increases the risk of vulnerabilities and, thus, potential attack vectors [176]. Moreover, increased connectivity broadens the attack surface with a higher potential to expose vulnerabilities. Since a modern vehicle can contain over 100M lines of code, aligned with a rough estimation of at least one bug per 1000 lines of code, it indicates more than 100k bugs in a modern vehicle. Moreover, in [32], T. Llanso and M. McNeil estimate that at least 1% of software vulnerabilities can be exploited, further indicating around 1k potential ways to compromise the vehicle software. Thus, due to technological advancements such as increased connectivity and the introduction of autonomous driving, incidents that require digital forensic investigations will inevitably rise to become more prevalent in the future.

Incidents can be intentionally caused by targeted cyber attacks and non-intentionally due to, e.g., a distracted driver or a software or hardware failure. Moreover, as shown in [22–24, 177], cyber attacks can be associated with life-threatening hazards due to their potential to affect safety-critical systems such as braking, steering, and engine control. For instance, in 2015, Charlie Miller and Chris Valasek hacked a Jeep Cherokee remotely over the Internet [22]. Although the safety-critical systems were isolated, a control unit had access to the communication bus and was vulnerable to reprogramming. Thus, the hackers managed to add code to the control unit and use it to send arbitrary CAN signals over the Internet to control, e.g., brakes and steering. In [24], Karl Koscher et al. demonstrated the potential to extract and reverse-engineer firmware to understand hardware features, which enabled them to add new functionalities and malicious code to a telematics unit. Moreover, the malicious code automatically erased any evidence of its existence after a crash. Thus, there was no post-incident available data for an investigation.

In 2018, a woman was killed by an autonomous vehicle [25]. The software did not correctly identify the individual as a pedestrian. In [26], a driver was using the autopilot and crashed into a vehicle in front. Afterward, vehicle data, such as, information from sensors, was used in digital forensics investigations. Establishing incident traceability regarding the driver, the autopilot, and potential threat actors is imperative.

In [178], six threat actors are identified, namely, the Financial Actor (FA), the Foreign Country (FC), the Cyber Terrorist (CT), the Insider (IN), the Hacktivist (HA), and the Script Kiddie (SK). For instance, the FC, CT, and HA can use one or many vehicles as moving weapons targeting humans or buildings. The FA might install ransomware that disables the ignition until a ransom is paid. The IN might add backdoors in vehicle software, while the SK can execute others developed exploits, usually with an unclear agenda. Beyond cyber attacks, the owner might manipulate the vehicle to gain more functionality, such as chip tuning, avoiding route tracing, and changing the odometer. Cases and incidents, as previously described, are highly relevant to identify and trace in automotive digital forensic investigations.

Current regulations, such as the UN R.155, require that vehicles provide data forensic capability for analyzing attempted or successful cyber attacks [8]. However, no details are provided on how to fulfill these legal requirements, and current vehicles have limited capabilities for enabling digital forensic investigations in cybersecurity-relevant incidents.

Failures in hardware, software, and cyber security issues must be possible to detect and investigate. Thus, a forensic-enabled vehicle needs to support mechanisms that generate, store, and secure forensically relevant data and differentiate between different types of malfunction.

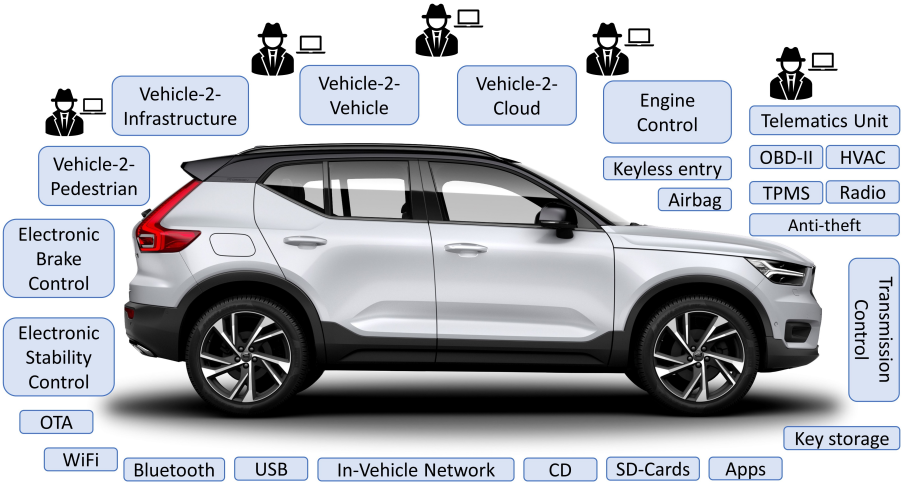


Figure 5.2: Example of potential attack vectors

5.1.4 Goal

Problem. We believe that the lack of digital forensics guidelines and digital forensics mechanisms within the automotive industry is a valid concern. To our best knowledge, no previous Systematic Literature Review (SLR) has been done within this field, that identifies the current work, and identify, categorize and map forensically relevant information to security properties and data users. The paper aims to answer the following questions:

- *What research exists within the field of automotive digital forensics?*
- *What is the coverage and specificity in different databases for automotive forensics search queries?*
- *What technical solutions exist with regard to automotive digital evidence, and how do these solutions uphold security properties?*
- *What forensically relevant data can be derived from existing literature and who are the stakeholders for this data?*

Approach. We have performed an SLR over work published between 2006 - 2021 within the field of automotive digital forensics and grouped them into two core categories, namely *technical solutions* and *surveys*. Another 11 categories, some with sub-categories, were identified from the selected work based on focus areas (cf. Table 5.1 and 5.2). We have identified gaps and discussed issues and challenges with respect to these categories. Moreover, we stated if any security properties were considered in the proposed *technical solutions*.

From the result of the SLR, we identified additional categories (cf. Table 5.3), this time specifically concerning forensic data, which were further mapped to required security properties and potential stakeholders. The aim was to identify data to be considered for automotive digital forensic investigations and ensure that data is reliable and secured.

An SLR is a recognized and standardized approach to provide broad coverage over publications concerning a particular field of interest. By following well-established processes, we provide confidence that other practitioners and researchers in the area do not need to repeat this work for the same period of time. Still, we give enough details to enable replicating the approach for a future time span to follow the progress within the field.

Contributions. Our main contributions are:

- We have performed an SLR within the field of automotive digital forensics.
- We performed database searches in four of the largest databases with the aim to get broader coverage and to investigate the individual coverage and specificity for each database. Backward and forward snowballing was performed on the selected work to increase the coverage even more.
- We have identified categories based on focus areas in the selected work and mapped the technical solutions in this work to the security properties which are considered.
- We identified and categorized forensically relevant data and mapped this data to potential stakeholders.

- We have also identified and discussed challenges, issues and research gaps within the area of automotive digital forensics.

We believe our contributions can be used as the basis for incorporating forensics into vehicle design for stakeholders such as automakers and law enforcement agencies. We also believe that our contributions can guide both performing automotive digital forensic investigations and encourage more research within this area.

The remainder of this paper is organized as follows: In Section 5.2, we list and explain requirements and the security properties used, Section 5.3 lists stakeholders for automotive digital forensic, and Section 5.4 presents related work. Section 5.5 details our approach and presents a comprehensive list of further categorized and mapped elements based on automotive forensic relevance. Section 5.6 presents the identified data categories mapped to security properties and data users, followed by a discussion of the result in Section 5.7. We end the paper with the conclusion in Section 5.8.

5.2 Requirements and Security Properties

A forensically enabled vehicle must fulfill basic security requirements and support techniques such as secure data logging and secure data storage [178]. A forensic investigation requires trust in the chain of events, such as the logical order of braking, acceleration, and steering. Moreover, digital forensics has strong dependencies on information security to ensure trustable data. As mentioned in Section 5.1.3, a forensics investigation includes the following basic steps, here further elaborated with relevant questions.

Identification. What is the reason for the incident? What data is relevant, and where is the data stored? What resources, e.g., tools and subject matter experts, are needed?

Preservation. How can we preserve integrity during data collection? For instance, for running devices and devices with remote access capabilities. Can the devices be turned off without losing data? Can data be remotely changed or erased?

Acquisition and verification. How can we extract the data (e.g., creating images and performing live acquisition)? How can we validate the authenticity of the data (e.g., with signatures and hashes)?

Analysis. What type of information is relevant to assess?

Reporting. How can we document all parts of the forensic investigation process?

Requirements to ensure admissibility in legal proceedings for vehicle forensic data regarding the fulfillment of security properties are stated in works, such as [179–181]. Thus, in line with these requirements, and shown in Figure 5.3, we adopt the well-known *CIA* security triad extended with two other properties and consider the first four as prerequisites for securing vehicle forensic data and the fifth for personal data, namely:

- *Confidentiality (C)*

- *Integrity/Authenticity (I)*

- *Availability (A)*

- *Non-Repudiation (N)*

- *Privacy (P)*

Confidentiality ensures that only authorized entities can access and disclose data. *Privacy* is related to personal data, such as traffic violations, location data, and synced data from external devices (e.g., text messages, calender's and phone records). Thus, there is a need to protect such data according to local laws and regulations [9,182,183]. *Authenticity* is a form of *integrity* that ensures data origin and is of particular interest for forensic investigations. *Availability* of data should be ensured, e.g., in the event of a crash and secure and tamper-proof storage guaranteed. *Non-Repudiation* ensures that the occurrence of an event and its origin can not be denied. Thus both *authenticity* and *integrity* are prerequisites for *Non-Repudiation*.

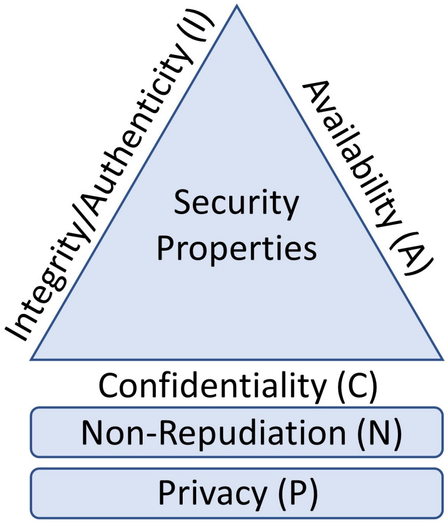


Figure 5.3: Visualization of the considered security properties

5.3 Stakeholders

We have chosen the four most common data users referred to in the literature [179, 180, 184–186].

- *Law Enforcement (LE)*

- *Vehicle Manufacturers (VM)*

- *Vehicle Drivers (VD)*

- *Insurance Companies (IC)*

LE such as the police and the related legal system requires reliable data to make a case and for the data to be admissible in a court of law. *VMs* need to have fault tracing data to distinguish between software and hardware failures and cyber security issues, e.g., fixing bugs and releasing software update patches. *VD* may manipulate forensic data, e.g., to hide, remove or manipulate digital evidence. *ICs* are interested in insurance cases and accidents and cost/risk policies for driver behavior (e.g., driver profiling).

5.4 Related Work

In 2004, the National Institute of Standards and Technology (NIST) published SP 800-72 for Personal Digital Assistant (PDA) forensics [187] of PDAs such as Pocket PC, Palm OS, and Linux based PDAs. Currently, PDAs have to a large extent, been replaced by other technologies such as smartphones and apps. In 2006, NIST released SP 800-86, a document for practical guidance on performing computer and network forensics [188]. SP 800-86 defines digital forensics as applying science to the identification, collection, examination, and analysis of data while preserving the integrity of the information and maintaining a strict chain of custody for the data. Another standard for digital forensics, the ISO 27037, was established in 2012 further reviewed and confirmed in 2018 [189]. The ISO 27037 provides guidelines for identifying, collecting, acquiring, and preserving digital evidence. In 2014 SP 800-101r1 was released for mobile device forensics providing guidelines for tool usage and procedures [190].

As mentioned in the introduction (cf. Section 5.1.2), another related area is IoT forensics which focuses on devices with Internet capabilities, including smartphones. For example, in [172], Stoyanova et al. list challenges, approaches, and open issues within IoT forensics. Although we can find similarities between IoT/mobile and automotive digital forensics (e.g., embedded devices with limited performance), automotive forensics is a comparatively different area, given the complexity of IVNs (cf. Section 5.1.1) and safety-critical requirements.

Previous work within automotive digital forensics (cf. Table 5.1 and 5.2) briefly mention the field of automotive digital forensics, such as issues and challenges. However, to our best knowledge, there has so far not been any systematic literature review that offers the comprehensiveness of present work. We have assessed over 300 papers and contribute with a comprehensive categorization and overview specific to the digital automotive forensics landscape based on focus areas, forensic data, security properties, and stakeholders. In the absence of standardized methods or guidelines for automotive digital forensic investigations, guidelines that consider the complexity of the vehicle architecture are of vital importance.

5.5 A systematic Literature Review

5.5.1 Approach

We have performed a systematic literature review based on the approach visualized in Figure 5.4.

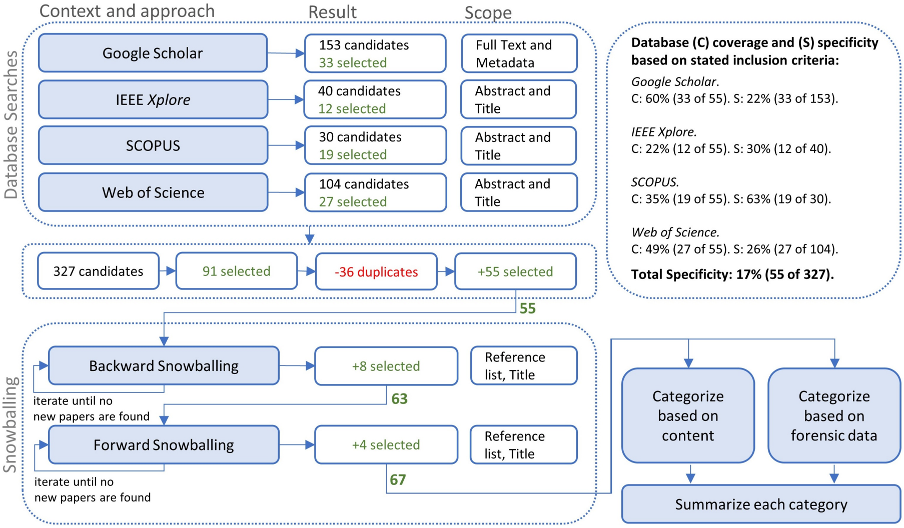


Figure 5.4: The process of our approach, result and scope

In total, 327 papers were acquired from the searches. Each individual database search was assessed further in a screening process by first reading the title followed by the abstract, conclusion, and for potentially relevant papers also skimming through the whole text in the article. Papers were included based on:

i. *relevance to the automotive domain.*

ii. *papers published between 2006 - 2021.*

iii. *papers published in journals and conferences.*

We have excluded articles:

i. *not specific to automotive digital forensics.*

ii. *not written in English.*

We have used the following search terms: *forensics* in conjunction with *vehicle*, *car*, or *automotive*. As shown in Figure 5.4, *Google scholar*[1] resulted in 153 candidates, where 33 papers were selected. *IEEE Xplore*[2] resulted in 40 candidates, where 12 papers were selected. *Scopus*[3] resulted in 30 candidates, where 19 were selected. *Web of Science*[4] resulted in 104 candidates, where 27 papers were selected. Figure 5.4 shows that the screening process resulted in 91 papers. After removing 36 duplicate papers, 55 papers remained. Figure 5.5 shows that some of the selected papers were present in more than one database where, e.g., *SCOPUS*, *IEEE Xplore*, and *Web of Science* have three of the selected papers in common [185, 191, 192]. Only two selected papers were found

[1] https://scholar.google.com/ search date: 2021-02-16
[2] https://ieeexplore.ieee.org/ search date: 2021-03-22
[3] https://www.scopus.com/ search date: 2021-03-24
[4] https://www.webofscience.com/ search date: 2021-03-29

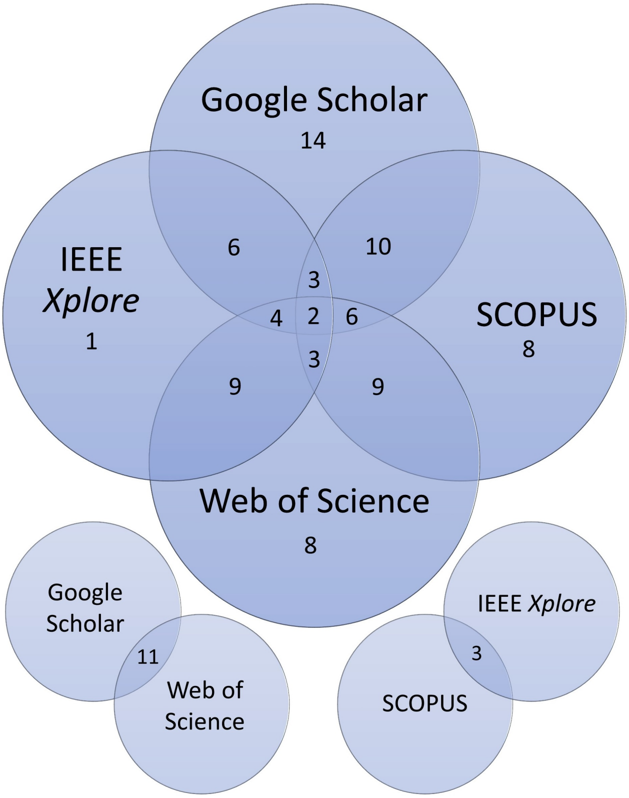


Figure 5.5: Overlaps and uniqueness for the selected papers

in all four databases [185, 191]. The selected papers that are unique and can only be found in one of the four databases are also shown where, e.g., *Google Scholar* has 14 unique papers and *IEEE Xplore* only one [193]. Figures 5.4 and 5.5 are further discussed in Section 8.5.

Snowballing. We then performed a snowballing approach [194]. As shown in Figure 5.4, we first performed *Backward Snowballing* where we extracted all references from the previous result and used the same search criteria as before with regard to the title and venue. Duplicates were removed, and another eight papers were found. We then performed *Forward Snowballing* with *Google Scholar*, based on those papers that cite any one of the previous 63 papers. The same search criteria were used once more, and another four papers were found; and as shown in Figure 5.4 and listed in Table 5.1 and 5.2, 67 papers were selected in total.

5.5.2 Categorization of papers

We have divided the results from the SLR into two main categories: *technical solutions* and *surveys*. Articles that propose technical solutions are included in the former category, and all others are included in the latter. Furthermore, all papers are mapped to one or more of the below 11 categories that were identified during the SLR based on focus areas we could identify when reading

Table 5.1: Selected papers concerning technical solutions

	**Retrieved from Snowballing* *(Co)Conference* *(Jo)Journal*		*(C)Confidentiality* *(I)Integrity* *(A)Availability* *(N)NonRepudiation* *(P)Privacy*																			
Ref.	**Author**	**Publ. Year**	**Details**	**1a. Data:** Data Collection	**1b. Data:** Extract. techniques	**2a. Challenges:** General	**2b. Challenges:** Req./Guidlines	**3a. Com.:** Cloud/Fog/Edge	**3b. Com.:** VANETs	**4a. Software:** In-vehicle	**4b. Software:** Tools	**5a. Hardware:** Architecture	**5b. Hardware:** Sensor	**5c. Hardware:** EDR/Blackbox	**6a. Alg.:** Machine learning	**6b. Alg.:** Other alg.	**7a. Cryptography:** Blockchain	**7b. Cryptography:** Other crypt.	**8. Framework and Processes**	**9. Practical Experiments**	**10. Infrastruct./Smart Cities**	**11. TEE/Virtualization**
[195]	C. Oham et al.	2021	Jo. I.A.N.P.									●					●	●	●	●		
[196	C. Yoon et al.	2021	Jo. I.P.					●					●				●		●			
[197]	C. Alexakos et al.	2021	Jo. I.	●		●	●				●	●							●			
[198	M. Waltereit et al.	2021	Co.	●									●	●					●	●		
[199]	P.A. Abhay et al.*	2021	Jo. C.I.							●			●	●			●		●			
[200	M.A. Hoque et al.*	2021	Co. C.I.			●												●	●			
[201]	J. Daily et al.	2020	Jo. I.	●	●						●			●					●	●		
[202	P. Sharma et al.	2020	Co.									●	●		●				●			
[203]	H. Guo et al.	2020	Jo. I.	●										●		●	●					
[204	A. Philip et al.	2020	T.Jo. I.P.												●		●		●			
[205]	M. Li et al.	2020	Jo. C.I.A.N.P.	●			●									●	●	●	●			
[206	Z. Ma et al.	2020	Jo. C.I.P.						●				●			●		●				
[180]	N. Vinzenz et al.	2020	Jo. C.I.A.N.P.	●			●					●		●				●				
[207	A. Mehrish et al.	2020	Jo.							●			●		●	●						
[208]	M. Waltereit et al.	2019	Co.			●										●						
[209	K. Bahirat et al.	2019	Co.	●			●						●		●							
[210]	L. Cintron et al.	2019	Co. I.N.P.	●												●	●		●		●	●
[211	S. Lee et al.	2019	Jo. I.N.				●					●		●		●		●				●
[212]	L. Davi et al.	2019	Jo. I.N.									●					●		●			
[213	D. Billard et al.	2019	Co. N.P.										●				●			●		
[179]	X. Feng et al.	2019	Co. C.I.A.P.	●		●	●											●		●	●	
[193	X. Wang et al.	2019	Co.					●								●				●	●	
[214]	M. Ugwu et al.	2018	Jo. I.N.				●							●			●	●	●	●		
[215	M. Marchetti et al.	2018	Jo.	●		●										●						
[216]	H. Guo et al.	2018	Co. I.N.	●										●			●	●				
[217	R. Hussain et al.	2018	Jo. C.I.N.P.					●	●				●					●	●			
[184]	M. Cebe et al.	2018	Jo. I.N.P.	●		●	●										●	●	●			
[192	M. Hossain et al.	2017	Co. C.I.	●		●						●				●	●		●	●		●
[218]	A. Mehrish et al.	2017	Co.										●			●						
[219	X. Feng et al.*	2017	Co. C.I.															●	●		●	
[185]	H. Mansor et al.	2016	Co. C.I.A.P.	●			●	●		●												
[220	A.D. Sathe et al.	2016	Co.	●						●			●									
[221]	N.Watthanawisuth et al.	2012	Co.	●									●	●						●		
[222	D. Nilsson et al.	2008	Jo. I.N.	●			●												●			

the papers. We specify whether the papers are published in a conference or journal and what security properties are considered concerning the proposed technical solutions.

In the remainder of this section, we have picked the most representative studies for each category and refer to Table 5.1 and 5.2 for the complete list.

1a. Data: Data Collection.

Papers in this category discuss the different types of forensic data and retrieval of such.

A. Attenberger [223] presents considerations for data generated in vehicles and categorizes these into two groups, namely, *front end* and *back end.* The former are vehicle electronics inside the vehicle, such as the infotainment module, and the latter, outside the vehicle, such as the cloud. A significant challenge for the *front end* is that there are no standardized interfaces for information extraction and no standardized format for storage. Additionally, in some cases, debug ports are lacking; thus, it is necessary to remove storage circuits for further data handling. Moreover, considering the steady increase of

Table 5.2: Selected papers concerning surveys

**Retrieved from Snowballing*

(Co)Conference
(Jo)Journal

Ref.	Author	Publ. Year		1a. **Data:** Data Collection	1b. **Data:** Extract. techniques	2a. **Challenges:** General	2b. **Challenges:** Req./Guidlines	3a. **Com.:** Cloud/Fog/Edge	3b. **Com.:** VANETs	4a. **Software:** In-vehicle	4b. **Software:** Tools	5a. **Hardware:** Architecture	5b. **Hardware:** Sensor	5c. **Hardware:** EDR/Blackbox	6a. **Alg.:** Machine learning	6b. **Alg.:** Other alg.	7a. **Cryptography:** Blockchain	7b. **Cryptography:** Other crypt.	**8. Framework and Processes**	**9. Practical Experiments**	**10. Infrastruct./Smart Cities**	**11. TEE/Virtualization**
[186]	K. Buquerin et al.*	2021	Jo.	●		●					●								●	●		
[223	A. Attenberger et al.	2020	Jo.	●		●									●							
[224]	R. Rak et al.	2020	Jo.	●		●								●								
[225	D. Kopenkova et al.	2020	Co.	●		●								●								
[226]	H.S. Lallie	2020	Jo.	●		●					●		●							●		
[227	K. Dološ et al.	2020	Jo.												●	●			●	●		
[228]	N. Le-Khac et al.	2020	Jo.	●	●	●					●									●		
[229	D. Steiner et al.	2019	Co.	●	●															●		
[230]	D. Sladović et al.	2019	Co.	●	●	●					●								●	●		
[231	N. Vinzenz et al.	2019	Co.	●										●						●		
[232]	M. Hussain et al.	2019	Jo.			●		●												●	●	
[233	C. Urquhart et al.	2019	Jo.	●		●														●		
[234]	C.J. Whelan et al.	2018	Jo.	●							●									●		
[235	A. Koch et al.	2018	Co.	●		●	●													●		
[236]	S. Tatjana et al.	2018	Co.	●							●		●	●						●		
[237	F. Leuzzi et al.	2018	Co.	●		●	●							●	●				●			
[238]	I. Cvitić et al.*	2018	Jo.	●	●																	
[181	C. Huang et al.	2017	Jo.			●		●				●								●		
[239]	Z.A. Baig et al.	2017	Jo.	●		●		●												●	●	
[191	D. Jacobs et al.	2017	Co.	●	●	●														●		
[240]	R. Altschaffel et al.*	2017	Co.			●	●				●			●					●			
[241	W. Bortles et al.*	2017	Co.	●							●		●	●					●	●		
[242]	J. Lacroix et al.	2016	Jo.	●	●				●		●	●								●		
[243	J.S. Ogden et al.	2016	Jo.	●	●						●			●					●	●		
[244]	N.Krishnamurthy et al.	2014	Jo.										●							●		
[245	D.W. Park et al.	2014	Jo.	●										●						●		
[246]	K. Lim et al.	2014	Jo.	●									●									
[247	J. Johnson et al.*	2014	Co.		●		●				●			●				●	●			
[248]	T. Hoppe et al.*	2012	Jo.								●		●						●	●		
[249	S. Al-Kuwari et al.*	2010	Co.									●	●									
[250]	D. Nilsson et al.	2008	Co.				●												●			
[251	J. Daily et al.*	2008	Co.	●							●			●					●			
[252]	D. Nilsson et al.*	2008	Co.	●															●			

relevant data, manual approaches become infeasible.

Vinzenz et al. [231] investigate data storage in vehicles and analyze its significance. Four data types were identified, airbag Event Data Recorder (EDR), Electronic Control Unit (ECU), Telematic Platform, and Infotainment System, where the EDR data was emphasised.

Utilized by a case study on a 2016 Mercedes Benz E-Class model, D. Steiner et al. investigate the communication in a modern vehicle and the forensically relevant data artifacts that can be retrieved [229]. Essential data and communication endpoints were identified.

L. Cintron et al. [210] model a transportation event data collection system as a *Hyperledger Fabric blockchain Network* which is simulated in a virtual transportation environment. Available accident data is collected from other vehicles and roadside units. An open-source framework tested in a production environment with community and commercial support is highlighted as beneficial in their solution. However, their solution was simulated as a limited environment with stated assumptions and not tested in a natural setting that most likely does not adhere to these expectations. Several benefits, drawbacks, challenges, security, and privacy considerations for their solution are mentioned, e.g., it is beneficial that all identities in a distributed ledger network are authenticated and thus accountable for their action. At the same time, this raises privacy concerns. Storage is another issue since the ledger increases in size quickly. There are also reliability concerns because used components still are immature, as well as requirements for high-throughput that might not be fulfilled.

1b. Data: Extraction Techniques. *A category that looks into techniques for extraction of automotive digital forensic data from IVN memory storage.*

In [201], J. Daily et al. presents a data extraction technique for non-volatile memory when standard extraction via vehicle diagnostic tools is not possible due to, e.g., damaged electronic control modules. The in-circuit debug port (JTAG) is suggested as a non-destructive data extraction method. However, manufacturers sometimes close these ports for production vehicles to protect against cyber attacks; thus, the assumption, in this case, is that the port remains open when vehicles reach the market. The first step is to extract the complete image as raw binary data or decoded data to another workable destination, such as a surrogate that is not damaged. Two different tools for data extraction are tested, i.e., the Alientech KTag [253] and the PEmicro Cyclone [254]. The extracted images from the two tools were compared by calculating an SHA-256 hash, and the result was a match with the conclusion that any one of the two tools is acceptable to use. The next step was to extract specific events from decoded data, e.g., sudden deceleration. The performed data extraction is valid for the actual hardware used and might not apply to all hardware.

In [230], D. Sladović et al. describe the digital forensic stages performed within an investigation, the type of information extracted from the vehicle, and the extraction process. Three connection points are mentioned: the OBD-II port, directly to an ECU, and directly to the EPROM (requires disassembly). Finally, the infotainment system is discussed as containing the most useful information. The Berla iVe [255] is the recommended tool for data extraction from infotainment units. However, something to consider before purchasing tools is to validate brand compatibility.

2a. Challenges: General Challenges.

A category that considers automotive digital challenges of a more general nature.

In [224, 225], Rak et al. provide an introduction to automotive digital forensics and provide examples of a few data sources, e.g., EDR, telematics, keyfobs, ECUs, and cameras. A few issues are mentioned, e.g., the high development pace aligned with the strive to gain an advantage over competitors results in a lack of security measures. Thus, vehicles become more vulnerable to hacker attacks. Moreover, the lack of standardization in, e.g., data interfaces, recording units, data storage, and lack of unified approaches for the digital forensics process makes different manufacturers use their own developed strategies. Thus, making forensic investigation challenging due to variations between brands and vehicle models.

In [191], D. Jacobs et al. focus on that vehicles have various standalone computing devices, lack of data security, difficulties in extracting data, lack of guidelines and tools for vehicle forensics, and problems obtaining proprietary information from manufacturers. Similarly, in [228], Le-Khac et al. highlight that data in vehicles is spread in a distributed system in various locations, which requires extensive manual work to find and extract relevant data for automotive digital forensics investigations. Additionally, data needed to, e.g., connect a driver to a crime is in many cases not sufficient for a majority of existing vehicles. In contrast, a modern smart car contains a vast amount of valuable data, but there is still no available process or framework to guide automotive digital investigations in this case. Moreover, very few automotive forensic data extraction tools exist, e.g., Berla iVe [255] for infotainment and telematics systems limited to a few brands. Aligned with [191, 224, 225], the lack of security and forensic mechanisms and a framework to guide the forensic process are identified as challenges.

In [235], A. Koch et al. emphasize on data collection and its privacy-related challenges, such as performing privacy evaluation of all data and adhering to the existing laws and regulations. The importance of user transparency regarding data stored, transmitted, and processed is highlighted. Three data streams for data collections are identified, namely main memory, mass storage, and communication. The management of the increasing amount of data and how to retrieve data from these data streams are mentioned as challenges.

In [240], R. Altschaffel et al. discuss challenges such as the absence of openly discussed automotive forensic processes within the scientific community, lack of standardized components, inaccessible memory due to security measures to guard intellectual property, low storage capacity for in-vehicle devices, and no authentication of messages for in-vehicle communication. They mention that the reconstruction of previous events must follow scientific and well-proven principles to preserve the authenticity and integrity of the data and highlight the importance of the investigator not being affected too much by a starting hypothesis, something which can lead to bias. Existing solutions are rare, often isolated, and limited by secrecy and intellectual properties. In [242], J. Lacroix et al. mention challenges for digital forensics for VANETs, such as the continuous changes in network topologies, unreliable communication channels, and source tracing difficulties. They suggest using GPS data, vehicle cameras, and analysis of data remnants from infotainment system applications to trace and find perpetrators better.

2b. Challenges: Requirements, Guidelines.

A category that proposes solutions to make automotive forensic data admissible for digital forensic investigations.

N. Vinzent el al. [180] state requirements for securing vehicle forensic data, and propose to adapt the telematics unit to accommodate storage for a limited dataset by using a circular buffer (i.e., a *Last In First Out* approach). Furthermore, they state that their implementation only requires minor software changes and no additional hardware because they consider the latter an unrealistic requirement due to its associated costs. However, current vehicle hardware usually consists of the least required memory for the task at hand; thus, additional memory still has to be added (i.e., hardware changes) since it would otherwise consider a too small time span for a realistic amount of required forensic data. Moreover, the telematics unit, most likely by design, only has access to a fraction of the data since the telematics unit can belong to a separate domain for security reasons with limited data access. In our opinion, IVNs have to be adapted both in hardware and software to enable a realistic approach for storing available forensic data.

In [222] and [250], D. Nilsson and U. Larsson present a list of requirements for data collection and event reconstruction in three categories. First, requirements for detection and storage of security violations. Second, requirements to address the five forensics W questions: who (traceability for event), what (type of event), where (sender/receiver ID), when (time for start, duration, and end), and why (data/value content). Third, a list of hashes for all ECU firmware should be securely stored and accessible for comparison when extracting in-vehicle firmware to detect manipulation. We consider the first category as an imperative prerequisite for automotive digital forensics, which potentially can be solved by Intrusion Detection System (IDS) mechanisms. For the second category, security mechanisms such as cryptographic primitives for validating the authenticity of events can be considered. The last requirement relates to firmware update where mechanisms such as signed software and secure boot can mitigate undetected manipulation of software.

F. Leuzzi et al. provide an organizational framework of requirements for the traffic police to guide future research efforts [237]. The emphasis is on road events, such as traffic congestion, accidents, crimes, or natural disasters. Data from events can be found inside the vehicle, e.g., logs, and outside the vehicle, e.g., traffic data. Aligned with [223], machine learning is mentioned as important in future crime investigation. Automatic approaches to managing large volumes of data, alerting when data matches previous crimes, and predicting and finding preparations patterns for potential future crimes are identified as important in future research. Data from, e.g., number plate detection systems and locations from cellular devices' can create a database over the traffic flow to be used in data mining. The goal can vary, but a few examples are mentioned, such as linking a particular vehicle and individual to a specific location and time, aligned with opening and closing doors to determine the time window when an individual left and got back to the vehicle. Synced data between individuals' cellular devices and vehicle communication is an important source, e.g., phone calls, text messages, and calendar data. However, a significant challenge is privacy, where several organizations are still not compliant with the law. The lack of standardization for vehicles regarding data quantity, quality, and formats makes data retrieval problematic.

3a. Communication: Cloud/Fog/Edge.

Cloud, fog, and edge node communication concerning digital automotive forensic data is the focus of this category.

In [185], Mansor et al. suggest a mechanism that enables data collection and transfer to the cloud via smartphones. The phone is proposed to be connected to the OBD-II port via a Bluetooth or WiFi interface. However, exposing a Bluetooth or WiFi interface to the OBD-II port to communicate with the phone can potentially create a bridge between the vehicle and the Internet, thus being considered a cyber security risk. The OBD-II facilitates the execution and tracking of sensitive diagnostic commands. Various vulnerabilities and exploits related to the OBD-II connection have been found in the past, e.g., [120, 138, 142, 149, 151, 154, 157, 159]. For instance, a user's smartphone can be compromised (e.g., via malicious applications), creating remote access for hackers.

Trust in external devices should be kept to a minimum, and storage and transfer to the cloud are better handled by internal mechanisms controlled by the vehicle manufacturer. Already existing phone applications developed by the vehicle manufacturers (e.g., VolvoOnCall [256], OnStar [257]) are better suited due to increased control and, thus, less risk. The vehicle and the application can use secure connections and communicate via trusted cloud sources, and application privileges can be separated and controlled according to architectural design decisions, such as by isolating safety-critical functions.

According to C. Huang et al., there is a trend to move away from cloud implementations in favor of fog/edge computing to save communication bandwidth [181]. Fog nodes can be situated near roads to collect, process, and store data. Thus, roadside units are a potential enabler for fog nodes as an extension of cloud servers with performance benefits in communication and data storage. Increased traffic control to improve safety and data for forensic investigations are potential use cases. However, fog computing is still in early development with many challenges, e.g., security and a large volume of data. C. Huang et al. elaborate that many millions of connected cars, with an average of 30 TB of produced data every day, clearly challenge storage, processing, and bandwidth capacity.

In [193], X. Wang et al. propose a scheme to speed up accident handling based on *Multi access Edge Computing (MEC)* to determine the liability in rear-end accidents. This scheme consists of a forensic model for data collection of driving information before and after a collision to establish a data chain for the vehicles involved in the accident. Vehicles are assumed to periodically upload data, such as position, speed, and acceleration, to an edge infrastructure that collect, process, and analyze data. A MEC infrastructure is suggested for accident-prone locations, such as intersections and parking lots. An evaluation metric based on the driver attention level is presented, and estimation of vehicles liabilities with regards to accidents. However, data regarding a few places is not enough; therefore, a broader approach covering various areas and traffic situations is necessary. Moreover, more work is needed to establish if the proposed liability scheme connected to a driver's attention accurately reflects the fact.

3b. Communication: VANETs.

A category that looks into automotive digital forensic solutions for Vehicular Ad hoc Networks.

In [206], Z. Ma et al. propose a digital watermark algorithm for VANETs, embedding location, timestamp, and forensic device data into a real-time vehicle accident photograph. Digital watermarking is a technique used to hide a data label in digital carriers such as images and multimedia to verify integrity and authentication. VANETs can be used to communicate traffic warning messages to surrounding vehicles, and photographs can add valuable information in addition to text messages. However, there are several challenges, such as ensuring trustworthiness and the privacy aspect of the communication, still the main goals in [206] are to ensure integrity, detect tampering and at the same time enhance user privacy in digital photographs. Additionally, an approach based on neural networks for vehicle license plate recognition and information gathering is proposed.

In [217], R. Hussain et al. introduce incentives-based vehicle witnesses, which utilizes moving vehicles and roadside units as witnesses to incidents.

The emphasis is on security, privacy, and the adoption of the proposed service. Cameras in roadside units and vehicles are assumed to collaborate and take pictures of their environment. Forensic data should be sent anonymously to the cloud, thus preserving privacy. However, although privacy might be ensured for the data source (e.g., a vehicle), data may include potentially sensitive information, such as videos and pictures of individuals, an issue also highlighted by H.S. Lallie [226].

4a. Software: Applications and Software. *A category that focuses on specific applications and software used to manage and communicate automotive digital evidence.*

In [220], A. Sathe et al. propose a *road safety and location monitoring system* by using a module in the vehicle to gather geographical coordinates and acceleration variations. The aim is to provide road condition updates. A three-axis accelerometer is used, and the authors suggest a correlation between sudden changes in these axes to confirm road disturbances. The location for these road disturbances is mapped to accident zones. A phone application is used to retrieve updates from the database. However, the work lacks the security and privacy aspects of their solution. Insecure HTTP communication is used for the communication, which leads to the potential for malicious actors to intercept privacy-sensitive data.

N. Watthanawisuth et al. [221] propose a similar approach using an accelerometer, GPS device, GSM module, and a microcontroller to detect and act on accidents and automatically send messages to appropriate recipients such as a family member or an emergency medical service. Although the title of their work indicates usability for vehicles, the practical tests performed, with a claimed high detection accuracy, were only done for bicycles, and the same deficit as in [220] applies; the security or privacy aspect is not discussed for their solution.

4b. Software: Forensic Tools.

A category that focuses on tools that might be applicable for automotive digital forensics.

J. Lacroix et al. [242] highlight that automotive forensics is an under-researched area. Privacy aspects have to be considered for IVN data due to its uniqueness concerning driver habits and actions. They discuss the type of data possible to extract, such as data dumps through OBD, USB, and JTAG ports. They examine a data dump from a truck infotainment system with tools such as *Forensics Toolkit (FTK)*, *Encase*, and *Autopsy* and present the extracted sensitive data. The data can be used to derive information about who previously has driven the vehicle, which can be helpful in forensic investigations.

In [236], S. Tatanja et al. investigates devices and tools for reading out data from vehicles. Many devices are mentioned, such as Bosch CDR500 (Crash Data Retrieval) for post-crash analysis and Bosch KTS540 for diagnostics and retrieving fault codes and reports. A case study of a Toyota Yaris previously involved in an accident was performed, where forensic data retrieval was performed with a Bosch CDR500 on the EDR in the vehicle.

In [228], Le-Khac et al. compare a few general forensic tools: *Encase*, *Accessdata Forensic ToolKit*, and *Xways Forensic*. The tools are compared based on the compatibility of different filesystems and non-structured memory dumps with varied results. Notably, none of the tools support the QNX filesystem, which is common in the automotive industry. There are very few automotive digital forensics-specific tools available, and as mentioned in [236], the *Bosch CDR* is discussed, and the *Berla iVe.*

5a. Hardware: Architecture.

A category around vehicle architectural design and its alignment with automotive digital forensics.

In [212], L. Davi et al. propose a blockchain architecture for ECUs that can be used as a blackbox. Their approach utilizes consensus algorithms to allow only agreed transactions to be added to the blockchain. However, safety-critical systems' real-time requirements make this approach less practical. The cost of accommodating an utterly new architecture where all ECUs can sign and validate messages has to be considered.

J. Lacroix et al. [242] briefly introduce the vehicle architecture, such as explaining internal components and communication busses. Especially, the CAN bus is highlighted as an essential source for live vehicle forensic data as it can contain relevant error messages. Moreover, the infotainment system holds valuable data, e.g., media content from external devices, internal logging, and localization data.

5b. Hardware: Sensor.

Automotive digital forensic solutions and mechanisms concerning various sensors such as GPS, LIDAR, and cameras are the focus of this category.

In [218] and [207], A. Mehrish et al. discuss the increased use of cameras by, e.g., law enforcement and private individuals and its relation to forensics. However, for videos produced by vehicle dashboard-mounted cameras to be admissible in the court of law, the video´s authenticity must be verified. The authors propose an algorithm to extract engine vibration characteristics from blur patterns in the video as a unique vehicle signature and claim an identification accuracy of 91.04 percent. The generated signature, together with other related forensic data, may help achieve a higher accuracy level. The captured video may contain information about unrelated individuals and vehicles; thus, privacy concerns shall be considered and addressed in such a case [9,182,183].

In [244], N. Krishnamurthy et al. presents a new area of research called audio-based vehicle verification, a field that can be useful within automotive digital forensics. Their work experiments on vehicle sound to verify whether a particular sound sample can be mapped to a specific vehicle. The sound from the engine and air-conditioner was shown practical and discriminative for these use cases. However, there are many challenges, such as that audio-based vehicle verification is sensitive to sound disturbances. Moreover, the privacy and security aspects are not mentioned; noise data collection (i.e., audio recordings) and storage of such data require these considerations.

5c. Hardware: EDR and Blackbox.

A category that discusses requirements for Event Data Recorders and Blackboxes concerning automotive digital forensics.

In [247], J. Johnson presents a review of the digital forensics concept, emphasizing on data recorders, such as EDRs, and cryptographic methods to ensure the integrity of digital evidence. The authors discuss the trustworthiness of extracted data using existing methods and propose recommendations to align with currently accepted standards for digital forensic soundness. Examples of such can be archiving network data and encrypting and hashing to conceal and detect tampering of data. Moreover, they mention that forensic soundness is defined by the methods of extracting, analyzing, and presenting digital evidence that must be performed in such a manner that the results can be used in legal proceedings with a high degree of confidence in their admissibility. However, today, data extraction concerning vehicles often uses tools unsuitable for digital forensics, thus lacking forensics soundness requirements. Current storage is often not resistant to tampering, and if encryption is used, it is still often too weak, and no integrity validation occurs.

6a. Algorithms: Machine Learning.

Papers in this category look into the automation of the automotive forensic process concerning data handling.

In [202], P. Sharma et al. propose a forensic investigation protocol utilizing a supervised deep neural network architecture for post-analysis of attacks targeting vehicle sensors. Anomalies in gathered data from IVN memory storage, together with an analysis of an accident, can reveal traces of attacks such as spoofing/jamming of sensors. However, it is stated that the investigator should use real-world sensor data for the analysis, but instead, a simulator is used for the data generation. Thus, it would have been interesting to see if their protocol could detect sensor manipulation within an actual vehicle. A more in-depth elaboration on the potential use cases for their approach is lacking, which would have justified the useability.

In [227], K. Dolos et al. use a freely available data set from the *Hacking and Countermeasure Research Lab* [258] to identify and classify drivers based on driving behavior. A hypothetical hit-and-run forensic scenario was used with three suspects, and a supervised learning algorithm was utilized to find the most likely suspects. However, due to the novelty of the area, additional work is needed before such driver identification methods can be used within a forensic context.

In [223], Attenberger et al. suggest Machine Learning to automate data handling and driver identification. Supervised classification can be used to train an algorithm into classes of data connected to individual drivers. These classes can then be used to determine who has been driving the vehicle. However, since no standardized exchangeable data format exists for the automotive, automated approaches such as these become challenging. The Cyber-investigation Analysis Standard Expression (CASE) is mentioned [259] as a possible standard for the automotive to use.

6b. Algorithms: Other Algorithms.

A category for proposal of specific algorithms for management of automotive digital forensic data.

In [215], M. Marchetti et al. highlight that there is no public specification for the proprietary data exchanged over IVNs; this limits researchers' potential to develop solutions that detect deviations. Thus, they propose *Reverse Engineering of Automotive Data Frames (READ)*, an algorithm to extract unknown CAN messages to identify and label CAN frames. The authors in [215] argue forensic usability due to the possibility of identifying deviation in time series for periodic messages (e.g., acceleration and steering), which can indicate sharp turns, acceleration, and braking, known as indices before potential accidents.

In [198] and [208], M. Waltereit et al. propose an approach to calculate the probability that suspects were present at a crime scene when GPS data is unavailable. The algorithm executes in an automated manner and saves time compared to other manual approaches. Further, they consider a hit-and-run accident as stated by Hoppe et al. [248], where some suspects without alibi claim their innocence. A route reconstruction within proximity to the incident area is suggested for the involved vehicles to absolve innocent suspects. In-vehicle data concerning driving behavior such as braking, acceleration patterns, and wheel speed is used as input to an algorithm for reconstruction and likelihood for specific routes and the probability for suspects' location.

7a. Cryptography: Blockchain.

A category that looks into papers that propose solutions utilizing blockchain technology.

In [196], a high-level traffic investigation framework for sensor data is proposed based on a decentralized identity distribution on blockchain, derived from case studies of accidents utilizing digital data. Due to its high-level abstraction, an industrial application of the proposed method seems infeasible. In [195], C. Oham et.al. propose a blockchain-based framework for securing the IVN and argue that using this framework keeps track of authorized historical operations (state changes) executed by vehicle ECUs, thus enabling traceability and identification of compromised ECUs. However, the framework only considers actions from the vehicle manufacturer, service technicians, and communication between the vehicle and roadside units, thus failing to provide complete event traceability. A more comprehensive range of potential events caused by other actors is needed.

In [203], H. Guo et al. propose an event recording system enabled by blockchain with a *Proof of Event* mechanism for vehicle networks. The aim is to provide trustable information about events based on an election algorithm that selects a leader/verifier in a blockchain network consisting of three participants: *accident*, *witness*, and *verifier*. A lead verifier (a vehicle or a roadside unit) is selected with the help of an election algorithm, and other vehicles can either be witnesses or other verifiers. The involved participants in the proximity of the accident will agree on the order of events according to a voting scheme and save these events into a blockchain. The authors implement a proof of concept prototype to demonstrate the feasibility of the proposed method. Given the diversity and complexity of IVN and its data [181], we believe the approach proposed by Guo et al. might only be applicable for a small fraction of the forensic data.

7b. Cryptography: Other Cryptography.

A category that discusses and proposes cryptographic primitives to secure automotive digital evidence.

In [200], M.A. Hoque et al. propose a solution named AVGuard for data collection and storage of forensics logs with the ability to verify data integrity. The integrity of the logs is verified by using a hash chain and a Bloom filter [260], and logs are encrypted to ensure confidentiality. Moreover, they demonstrate a proof-of-concept implementation of their approach. Proofs of the logs are suggested to be published on the web. However, securing the web's communication and storage is not discussed, and although integrity and confidentiality are considered for the data collection, they lack the privacy aspects for their solution.

8. Framework and Processes.

A category that discusses and proposes frameworks for the management of automotive digital evidence and simplification of automotive digital forensic processes.

In [204], A. Philip et al. propose a framework for road accidents and traffic violations based on deep learning and blockchain. An accident warning system for vehicles is established by considering road and climate conditions and driving patterns as parameters. The best parameters for specific traffic segments can be predicted, and warnings can be issued to vehicles from roadside units.

In [184], an approach for a permission-based blockchain framework is proposed for data collection of various types of data such as health data (e.g., from wearable devices) and automotive diagnostic. Their approach integrates the vehicular public key infrastructure (VPKI) into the blockchain to provide membership and privacy. M. Hossain et al. propose a vehicle data collection framework for distributed, decentralized, and mobile entities with secure storage [192]. Mechanisms to collect and store digital evidence and a specific algorithm for data integrity verification are proposed for evidence verification.

In [186], K. Buquerin et al. presents a potential automotive digital forensic process according to four phases: *forensic readiness*, *data acquisition*, *data analysis*, and *documentation*. The OBD-II port was used as an example for a connection point for data collection with Wireshark. A Packet Capture (PCAP) file and a hash of the file were stored for analysis, and finally, a report was generated.

In [197], C. Alexakos states various challenges for integrating digital forensics into the Internet of Vehicles (IoV) context. Examples are that there are no standardized data formats and a dynamic network topology, i.e., nodes are added and removed. Thus, the topology continuously changes in different vehicles models. Moreover, there is no open access due to intellectual properties and privacy concerns. The authors in [197] propose a forensic readiness tool that follows the digital forensic process model from Valjarevic et al. [261]. The tool is implemented by software into the nIoVe framework [262]. The tool's purpose is to collect forensically sound data, which enables reconstruction of events to be presented in a court of law and to learn, mitigate and predict future anomalies such as cyber attacks.

In [240], R. Altschaffel et al. suggest using a Desktop IT forensic process model from S. Kiltz [263] additionally to EDRs for the automotive domain, which consists of various investigation steps: first strategic and operational preparations, which consist of measures taken before and after incidents, followed by data gathering, data investigation, data analysis, and finally, the actual documentation of the complete investigation. Moreover, various tools and potential use cases are mentioned, as well as live and static data acquisition and forensic data acquisition from ECUs, sensors, and actuators.

In [248], T. Hoppe et al. introduce an overview of the digital forensics process and put this into an automotive context. They suggest using a data recorder that securely logs existing navigation data transmitted over the CAN bus. Data, such as route information (e.g., street names) aligned with other data (e.g., speed, start, or destination position), can be used to reconstruct vehicle routes for post-incident investigations and further used as indices to individuals' locations at the time of crimes. Information such as this can either connect or free someone from involvement in the incident.

9. Practical Experiments.

A category that focuses on different types of practical forensic experiments.

In [233], C. Urquhart et al. perform practical experiments to pinpoint vulnerabilities within a Scoda Octavia vRS and suggest vehicle components and data that can be considered for digital forensic investigations. Infotainment data (e.g., Bluetooth ID of paired devices, call logs, and pictures/thumbnails), GPS, ECU memory, and diagnostic messages are mentioned. C. Whelan et al. [234] present a study that investigates available forensics artifacts in two different infotainment systems: a *Uconnect* system and a *Toyota Extension Box*. The *iVe* tool from *Berla Corporation* was used for data acquisitions. *Uconnect* provided only location data and *Toyota Extension Box* extensive user-related information such as contacts, call logs, and location data.

Vinzenz et al. [231] have analyzed crash data from the NHTSA NASS CDS database retrieved in the proprietary EDRX-format from a Bosch EDR tool. Findings were that before the year 2000, only airbag deployment status was stored. After the year 2000, vehicle speed during crashes was included, and starting in the year 2005, additional data was added, such as engine throttle. Increasingly more data in the form of Diagnostic Trouble Code (DTC) could be found for upcoming years. In total, 28 different DTC types were identified. Privacy and security considerations are mentioned. However, it is not completely clear if the data is admissible in court due to privacy laws and the lack of security measures to guarantee trustable data.

Le-Khac et al. [228] perform two case studies on extracting vehicle data. For the first case, they extract data from an infotainment system. For the second case, they investigate communication traffic on GSM/3G/4G networks and discuss how potential sources of evidence can be intercepted (e.g., PCAP data, metadata, and call detail records (CDR)). In [191], D. Jacobs et al. present a case study on a Volkswagen Golf version 6, 2012 station wagon and discuss considerations to avoid potential data losses (e.g., do not start the vehicle).

10. Infrastructure/Smart Cities.

A category that look into infrastructure communication and smart cities with regard to automotive digital forensics.

X. Feng et al. [179] discuss Autonomous Vehicles (AVs), Smart Cities (SCs), and digital forensics with an emphasis on sensor data. They provide examples of forensic data handling issues such as no data integrity validation and often unprofessional data extraction compared to other forensic areas. The authors conclude that current automotive data forensic acquisition and analysis approaches are not acceptable concerning legal evidence. Moreover, they propose a mechanism for acquiring sensor data from AVs in SCs and securely uploading it to cloud storage.

In [239], Z.A. Baig et al. discuss the challenges of digital forensics in smart cars. The difficulty of proving the validity of and access to vehicle data and incorporating cars into the context of smart cities are examples of the challenges identified by the authors. A hypothetical use case of a reckless driver is presented with the interaction of other smart city entities followed by the forensic investigation process.

11. Trusted Execution Environments and Virtualization. *A category that focuses on solutions that secure automotive digital evidence using Trusted Execution Environments (TEEs) and virtualization.*

In [211], S. Lee et al. suggest an automotive data recording system named T-Box that executes inside a trusted execution environment to detect data manipulations such as deletion, replacement, replaying, and truncation. S. Lee et al. claim that the T-Box data can be used as digital forensic evidence. However, their approach is platform-dependent and fails to protect data against tampering before storage. Moreover, the T-box does not consider the confidentiality and privacy aspects of the stored data.

5.6 Categorizing and mapping forensic data to security properties and data users

Automotive digital forensics requires identifying, acquiring and analyzing data that potentially can be used as digital evidence. The relevancy of a particular data to a forensic investigation depends on the type of crime being investigated. In this work, we consider the following data types:

- *data at rest*: data stored in the memory of ECUs or other automotive modules.

- *data in transit*: data that flows over networks.

- *data executed*: by ECUs/modules leading to specific events, e.g., state changes.

As shown in Table 5.3, we have identified forensically relevant data from the literature and placed them into categories, and further associated them to the required *security properties* and the potential *data users* (cf. section 5.2 and 5.3).

We have elaborated on the required security properties concerning the various data categories. For instance, detected anomalies are most likely not related to any privacy (P.) sensitive information or requirement for the fulfillment of non-repudiation (N.). Detecting anomalies, e.g., by an anomaly-based intrusion detection system, usually do not include the need to prove source origin. However, the data stored related to anomaly events might contain sensitive information and must be trusted and available to forensic investigations, i.e., fulfill the C.I.A. properties.

Another example is the use of actuators, concerning, e.g., braking, steering, and throttle control, that can be privacy sensitive due to their connection to individual driving patterns. However, the signal that initiates the actuator response does not need the confidentiality (C.) property but must be authentic (I.) and secured against, e.g., replay attacks; thus, the signal source requires (N.). Therefore, we consider I.A.N.P. as enough for this data category. Although requirements need evaluation on a case-to-case basis, our mapping to security properties can indicate and guide the reasoning when securing forensic data.

The mapping of forensic data categories to data users is based on the potential value for stakeholders. For instance, resilience techniques, such as responses to malfunction and cyber security issues, are most likely to interest

Table 5.3: A mapping from data type to security properties and data users

Data Category and Reference	***Security Properties**	****Data Users**	**Example of data and source**
External devices. [184–186, 191, 192, 196, 223, 225, 228–230, 233, 234, 237, 241, 242, 264]	C.I.A.N.P.	LE.VM.VD.	*Cellular phones* may contain data from calendars, call logs, text messages, email communication, images, documents and location data. *USB memory* may contain documents and media files. *Remote keyless entry systems (RKE)* has information about VIN, time and date for use. *OBD-II dongles* has internal memory that may contain, e.g., logs with information about use.
Sensors. [179–181, 184, 191, 192, 198–200, 202–204, 207, 209–214, 217–220, 223, 225–231, 233–237, 239–242, 244, 248, 249, 264]	C.I.A.N.P.	LE.VM.VD.IC.	Information about speed, position (gps), temperature, airbag and object detection (from cameras, LIDAR and lasers).
Actuators. [196, 202, 235, 240, 242]	I.A.N.P.	LE.VM.IC.	Signals sent to initiate braking, steering and engine control (e.g., throttle).
ECUs. [179, 180, 185, 186, 191, 195, 204, 211, 212, 222, 223, 225, 231, 233, 235, 237, 239, 240, 242, 243, 250, 252]	C.I.A.N.P.	LE.VM.VD.IC.	Data storage, may contain information about operating mode(s), internal state and decisions recently made.
Software update events. [222, 233, 248, 250, 252]	C.I.A.N.	LE.VM.VD.	Software update logs, e.g., information about failed and successful installations, software version numbers and authorization attempts. Detected events with regard to software manipulation, e.g., caused by cyber attacks or the vehicle owner that tries to extend functionalities.
Security events: Diagnostics. [180, 184–186, 191, 201, 222, 226, 228, 231, 233, 237, 239, 243, 248, 250, 252]	C.I.A.N.	LE.VM.VD.	Events such as attempts to activate/deactivate firewalls and run privileged diagnostics commands.
Security events: Resilience tech. [214]	C.I.A.N.	LE.VM.	Executed resilience mechanisms (events) as a response to, e.g., a malfunction or cyber security issue. Examples of such events can be a system state change, reconfiguration, and migration [178].
Security events: Anomalies. [185, 186, 215, 239, 240, 250]	C.I.A.	LE.VM.	IDS software. Detected anomalies (events) in communication traffic, such as during a cyber attack, e.g., port scans, brute force and DoS. Events in forbidden situations, e.g., software updates during driving and maximum velocity during parking mode. Events connected to malfunctions, erroneous results and hardware failures.
Safety events. [179, 180, 184–186, 196, 198, 201, 210, 212, 215, 217, 220, 222, 223, 226, 229–231, 233–239, 241, 250, 265]	C.I.A.N.	LE.VM.VD.IC.	EDR crash data. Information about safety-critical events such as braking, acceleration, steering, engine control, airbag release, and seat belt traction/on/off. Indirect safety-related data such as warning messages about distracted driving and tired driver.
Normal events. [184, 204, 222, 223, 230, 234, 237, 241, 250]	I.A.N.P.	LE.VM.VD.IC.	Events such as opening/closing of doors and trunk. Turing on/off the engine and events with regard to activation/deactivation of alarms and locking/unlocking the vehicle. Information about fuel levels and consumption, and oil level and temperature.
Biometrics. [248]	C.I.A.N.P.	LE.VM.VD.IC.	Events from biometric driver identification such as face recognition, fingerprints, and voice.
Settings. [248]	C.I.A.N.P.	LE.VM.VD.IC.	Information about individual driver settings with regard to position of the seat, mirrors and driving mode (e.g., economic, sport and normal).
Vehicle-2-Vehicle. [179, 184, 192, 204, 206, 210, 217, 219, 222, 223, 228, 230, 234, 242, 266]	C.I.A.N.P.	LE.VM.VD.IC.	Communication within VANETs such as information about accidents, location and traffic conditions.
Vehicle-2-Infrastructure. [179, 184, 186, 191–193, 204, 206, 210, 217, 219, 222, 223, 228, 232, 234, 242, 250, 266]	C.I.A.N.P.	LE.VM.VD.IC.	Information from traffic lights and traffic regulations, e.g., speed, as well as detected violation of such. Exchanged traffic information, e.g., information about weather conditions, accidents and traffic jam, and route recommendations.
Vehicle-2-Pedestrian. [192, 204, 210, 228]	C.I.A.N.P.	LE.VM.VD.IC.	Information about location data (e.g., gps and gyrometer in relation to collision and rollover detection). Communication from vehicle to the relevant profession concerning accidents, e.g., vehicle to a physician or police.
Vehicle-2-Cloud. [179, 185, 192, 217, 223, 225, 227, 228, 232]	C.I.A.N.P	LE.VM.VD.IC.	Events from IDS/IPS, software updates and traffic information. Communication using applications (e.g., onStar [257] and VolvoOnCall [256]).

*(C)Confidentiality, (I)Integrity, (A)Availability, (N)Non-Repudiation and (P)Privacy. **(LE)Law enforcement, (VM)Vehicle Manufacture, (VD)Vehicle Driver and (IC)Insurance Company.

the vehicle manufacturer (VM.) and law enforcement (LE.); for the former to solve potential vulnerabilities and for the latter to investigate possible crimes. In contrast, the value of data from such events is minimal for the vehicle driver (VD.) and the insurance company (IC.).

We do not provide a complete list of data, and we know that there can be overlaps. Nevertheless, the identified data in Table 5.3, in conjunction with the categorization of the current work in Table 5.1 and 5.2 can help developers and architects to make informed decisions on data collection concerning automotive digital forensics.

5.7 Discussion

An interesting observation from Figures 5.4 and 5.5 is that research publications are distributed in various databases. Thus, combining many database searches with snowballing improves the coverage of the work. Google Scholar has the broadest coverage of the selected papers and, at the same time, the lowest specificity. Therefore, Google Scholar required the most effort compared to the other databases. SCOPUS required the least effort but at the cost of less coverage.

The low overlap was surprising and its potential consequence on coverage, e.g., without Google Scholar, the number of selected papers would have decreased by a quarter. IEEE Xplore seems redundant with only one unique publication. However, the impact of one paper can be significant both when it comes to contribution and later in the snowballing process; thus, we still consider the use of multiple databases critical for coverage.

Data collection is part of the digital forensic acquisition and, as shown in Table 5.1 and 5.2, is logically included in a majority of the studies. Most technical solutions consider C.I.A., but only two consider C.I.A.N.P., while some do not consider security properties at all. Blockchain is relatively common in technical solutions in comparison to, e.g., solutions utilizing Trusted Execution Environments (TEE) and virtualization. The latter two are both promising approaches due to their potential to isolate the execution of sensitive processes such as cryptographic operations, e.g., hash generation, signing, and encryption/decryption. Few solutions propose forensics tools and extraction techniques, which align with challenges such as no standardized data formats and no guidelines or standards for the automotive domain concerning automotive digital forensics. Automotive digital forensics is a rather immature area, and we believe that security and privacy need to be emphasized more in the automotive literature.

By adding forensics mechanisms to vehicles, such as increased data collection concerning individual driving patterns, biometrics, and whereabouts, the potential to solve crimes increases. However, we face the risk that a vast amount of sensitive data will be stored in various locations. As mentioned in the introduction (cf. Section 5.1.3), several potentially exploitable vulnerabilities in vehicles can be assumed, with the risk of compromising this data. Security mechanisms that protect such data need to be strengthened, and individual control of this data established. Still, there is always a risk that data can be misused even when the retrieval as such is authorized. For example, car rental or insurance companies can use data from, e.g., individual driving patterns to decide customers' prices for their services according to risk profiling. The ethical discussion of where to put the bar between data collection and the user's right to privacy is out of scope in this work and something for politicians and lawmakers to find common ground.

Lack of proposals for TEE and virtualization solutions, lack of usable tools, and lack of possibilities for data extraction require additional research effort. Essential observations from Table 5.3 are that data related to, e.g., *sensors*, *ECUs*, and *safety* are considered valuable forensic data. In contrast, data from *biometrics* and *settings* are only mentioned in one study. Security events from detected anomalies and executed resilience techniques are only considered in seven papers. Thus, more research on detecting, storing, and extracting forensically relevant events such as these are required.

Vehicle security mechanisms, e.g., IDS and firewalls, perform actions and can detect relevant events for digital forensic investigations. Examples of such events can be anomalies in traffic patterns during cyber attacks, successful and failed authentication attempts, and port scans. Such events must be securely stored together with information about the logical order w.r.t. to time. For instance, the airbag might be deployed due to a cyber attack causing a crash. Thus, establishing the order of events is imperative for the investigation to find the root cause of the crash. Moreover, abnormal activation of lane assist function, opening/closing doors, increased volume of music, and activated windscreen wiper before a fatal accident may provide digital evidence of a crime, e.g., an attack intended to distract the driver, potentially leading to an accident.

Data related to vehicle-to-everything (V2X) communication will continue to increase and, thus, be more prominent in future vehicles. Communication between vehicles and infrastructure, roadside units, other vehicles, and pedestrians can impact a specific accident. Therefore, location data and events between vehicles and pedestrians are relevant and important to synchronize to correlate and connect different incidents. Moreover, it is essential to differentiate between faults such as safety-related malfunctions (e.g., component failures) and cyber security issues.

5.8 Conclusion

We have performed a systematic literature review in the field of automotive digital forensics. We performed our searches in four major databases and performed backward and forward snowballing to maximize the coverage. Two core categories were identified for the selected work, namely *Technical Solutions* and *Surveys*. Additionally, 11 categories, some with sub-categories, were identified from the contents of the papers, categories to which all papers were then mapped. Moreover, technical solutions from the selected papers were linked to the security properties they cover. We have further identified and categorized relevant forensic data types derived from the selected papers and linked them to security properties and data users. We have identified and discussed challenges, issues, and research gaps within the area of automotive digital forensics.

The use of a well-known and standardized approach, SLR, gives confidence that the essential papers in the searched databases are found. Thus, it should not be necessary to repeat this work by practitioners or researchers to find relevant publications from this time period. Still, the SLR approach makes it possible to repeat the work in the future to follow the development of the area. The categorization gives a comprehensive overview of the forensics field and related research activities and makes it easy to find relevant papers in a particular sub-field of digital forensics. The number of papers in the categories also indicates what research areas have been considered important and challenging during the studied time period. The comparison of the search results from four large databases is interesting since it shows how much additional value searches in multiple databases may give. It is also useful to know to what extent they overlap and which ones to focus on when searching for publications since it might also apply to other areas. Our contributions are helpful not only for automotive digital forensics but also for similar systems, such as cyber-physical systems and smart cities.

In summary, our performed analysis guides further work within the area and benefits both researchers and practitioners. The identified forensic data categories can be used to indicate relevant data types to look for within investigations. The data categories can be used in conjunction with the identified technical solutions to serve as a guideline for implementing forensic mechanisms into vehicular and similar systems. Thus, it provides both tools and techniques for the data collection aligned with the data types to consider.

Chapter 6

UniSUF: A Unified Software Update Framework for Vehicles Utilizing Isolation Techniques

Format-adapted version that appeared in 19th escar Europe 2021

K. Strandberg, D. K. Oka, T. Olovsson

Abstract. Today's vehicles depend more and more on software, and can contain over 100M lines of code controlling many safety-critical functions, such as steering and brakes. Increased complexity in software inherently increases the number of bugs affecting vehicle safety-critical functions. Consequently, software updates need to be applied regularly. Current research around vehicle software update solutions is lacking necessary details for a versatile, unified and secure approach that covers various update scenarios, e.g., over-the-air, with a workshop computer, at factory production or using a diagnostic update tool. We propose UniSUF, a Unified Software Update Framework for Vehicles, well aligned with automotive industry stakeholders. All data needed for a complete software update is securely encapsulated into one single file. This vehicle unique file can be processed in multitudes of update scenarios and executed without any external connectivity since all data is inherently secured. To the best of our knowledge, this comprehensive, versatile and unified approach cannot be found in previous research and is a contribution to an essential requirement within the industry for handling the increasing complexity related to vehicle software updates.

6.1 Introduction

A vehicle can contain more than 150 ECUs (Electronic Control Units) and over 100M lines of code. The complexity of software within the automotive domain is increasing and with it the risk for vulnerabilities. To address this, there are ongoing activities for vehicle software updates, such as ISO/CD 24089 [11] and UN Regulation No. 156 regarding vehicle software update requirements [10]. The latter states, among other, requirements for vehicle manufactures to have a secure software update process. While standards and regulations typically focus on high-level requirements, technical design and implementation requirements are left up to the automotive organizations. There is a risk that if the software update process is vulnerable, it can be exploited by attackers who could potentially introduce malicious code at some stage into the software update process that finally reaches in-vehicle systems causing life-threatening hazards such as manipulated brakes, steering, or engine control.

A secure software update framework that can support numerous different update scenarios, such as over-the-air, in workshops, and in factories, with or without Internet access, is required for automotive organizations in order to apply software updates to address vulnerabilities in a timely and regular manner. Our approach is to provide a cost-effective, open architecture, with increased security through isolation and separation of duties that is comprehensive to support numerous use cases. Thus, we propose UniSUF, a versatile and unified approach for secure vehicle software updates. By using multiple signing and encryption keys, all data needed for a complete software update is securely encapsulated into one single file, the *Vehicle Unique Update Package (VUUP)*. This vehicle unique file can be processed by a vehicle ECU, using a workshop computer, at factory production or with a diagnostic update tool, hence considerably simplifying software management processes. At the receiving vehicle side, this file is decapsulated and validated layer by layer, where cryptographic material and sensitive operations are isolated within a trusted execution environment to ensure both the integrity and the confidentiality of the data. The main contributions of this paper are:

- We have analyzed and reviewed several software update use cases in the automotive industry and as a result, defined a number of constraints and conditions for a unified and versatile approach.
- Considering these constraints and conditions, we suggest an approach for vehicle software updates, well aligned with automotive industry stakeholders. In-depth details give a comprehensive overview for a possible secure implementation covering the whole software chain from producer to receiver.
- We have reviewed the suggested approach with automotive software update architects to ensure that the proposed approach can be practically deployed and efficiently adopted for vehicle software updates.

6.2 Problem Statement

Considering the different existing use cases for vehicle software updates, such as over-the-air, using a workshop computer, at factory production, or with a diagnostic update tool, each use case typically has its own approach which causes complexity. Moreover, new use cases for software updates need to be considered with future demands to support 3rd party component updates ([267], [268]). Therefore, to simplify, reduce costs, allow flexibility, and to make the update process manageable, all while considering security aspects, we propose a unified and versatile approach to handle all the use cases.

After reviewing the above-mentioned use cases, the following constraints and conditions are defined for a unified software update framework:

- Support for online updates (software update files and/or cryptographic credentials/operations require online access).

- Support for offline updates (software update files and cryptographic credentials/operations are accessible offline).

- Should not rely on additional input for cryptographic keys or installation instructions, e.g., from a diagnostic update tool (i.e., all data needed for a complete software update is securely encapsulated into one single file and no additional input is required).

- No dependency on the data distribution model (i.e., software update files can be provided through different means and it does not matter how they are distributed to the vehicle).

- No dependency on software update storage location (i.e., software update files should be independently protected regardless of where they are stored).

- Flexible and modular to support 3rd party component updates.

We have taken these constraints and conditions into consideration when designing a software update framework to allow for a unified and versatile approach to support different use cases. Our proposed software update framework is described in the next section.

6.3 UniSUF: A Unified Software Update Framework

In this section, we present the Unified Software Update Framework (UniSUF). First, an overview of the involved entities in the framework is presented, followed by a brief explanation on how to secure data distribution and data execution, and finally, the procedure for preparing software update files is given.

6.3.1 Entities

There are three main entities involved in the software update process: the producer, the consumer, and the repository. The producer is responsible for producing the software. The consumer is responsible for the download and installation process of the software, and the repository is a storage point for software preferably located in various cloud sources, enabling both proximity and redundancy for data in relation to the vehicle.

An overview of the data distribution in the backend handled by the ***Producer Agent (PA)*** is shown in Figure 6.1. The main entities it contains are:

- ***Producer Security Agent (PSA)*** facilitates functionalities for secure key generation using ***Secure Key Generator (SKG)***, secure storage for cryptographic material using ***Cryptographic Material Storage (CMS)*** and signing of data using ***Producer Signing Service (PSS)***.

- ***Version Control Manager (VCM)*** has control over available software versions w.r.t. current vehicle status to create both download instructions using ***Producer Download Agent (PDA)*** and installation instructions using the ***Producer Installation Agent (PIA)***.

On the receiving side, Figure 6.4 shows the ***Consumer Agent (CA)*** handling data distribution to the vehicle. The main entities it contains are:

- ***Consumer Download Agent (CDA)*** downloads required data, e.g., instructions and software files, verifies the authenticity of the data and initiates installation using the ***Consumer Installation Agent (CIA)***.

- ***CDA*** and ***CIA*** uses the ***Consumer Security Agent (CSA)*** which requires a Trusted Execution Environment (TEE) in order to support secure operations and store cryptographic keys securely.

By using isolation mechanisms and implementing each entity as a module according to the principles of *least privilege* and *separation of duties* a potentially compromised entity cause the least possible harm to the complete system. These modules can be secured either locally or in the cloud.

6.3.2 Securing Data Distribution and Data Execution

To be able to secure the *data distribution* and *data execution*, we propose using signed asymmetric and symmetric keys in conjunction with key wrapping mechanisms. Symmetric session keys are used to encrypt sensitive cryptographic material needed for the update processes, such as keys for unlocking ECUs and keys for decryption of software. The symmetric session keys are encrypted with a public vehicle unique asymmetric key, ensuring the secure storage and transfer of key material. Using an asymmetric key for key wrapping ensures that only the vehicle with the corresponding private key can decrypt the encrypted session keys.

Policies dictate rules for each individual encrypted session key, where policies and keys in conjunction are signed i.e., giving rise to a *Key Manifest (KM). KMs* are securely processed at the receiving side, where session keys are appointed to certain trusted applications according to the stated policies. The functionality of trusted applications can be decryption of software files, unlock ECUs for software updates, and signing of installation reports and logs.

6.3.3 Preparation of Software Update Files

The individual files that contain the actual software need to be secured, ensuring both confidentiality and authenticity. Considering the entities in the

framework the procedure to secure software files is as follows.
1. The producer of software signs software files with a supplier specific signing certificate to provide authenticity. If supported, this signature is later validated by end receivers (e.g., ECUs) before installation. Software files are uploaded to the *Producer Local Secure Storage*, shown in Figure 6.1.
2. *VCM* receives the software files and validates the software supplier's signature.
3. *VCM* requests a symmetric encryption key (hereafter called sw_key) from *PSA* and encrypts the software file with this key to provide confidentiality.
4. *VCM* requests a signature of the hash of the encrypted software file from *PSS*. The signature is added to the encrypted file metadata to provide authenticity.
5. *VCM* performs mutual authentication towards the cloud software repository and uploads the signed encrypted file to the cloud and stores the url to this file in a database.
6. *VCM* securely stores the symmetric encryption key (i.e., the corresponding sw_key) in *CMS* to be retrieved later, and encrypted and included into a *Secure Key Array (SKA)* for a future software update (cf. Step 6. in Section 4.1).

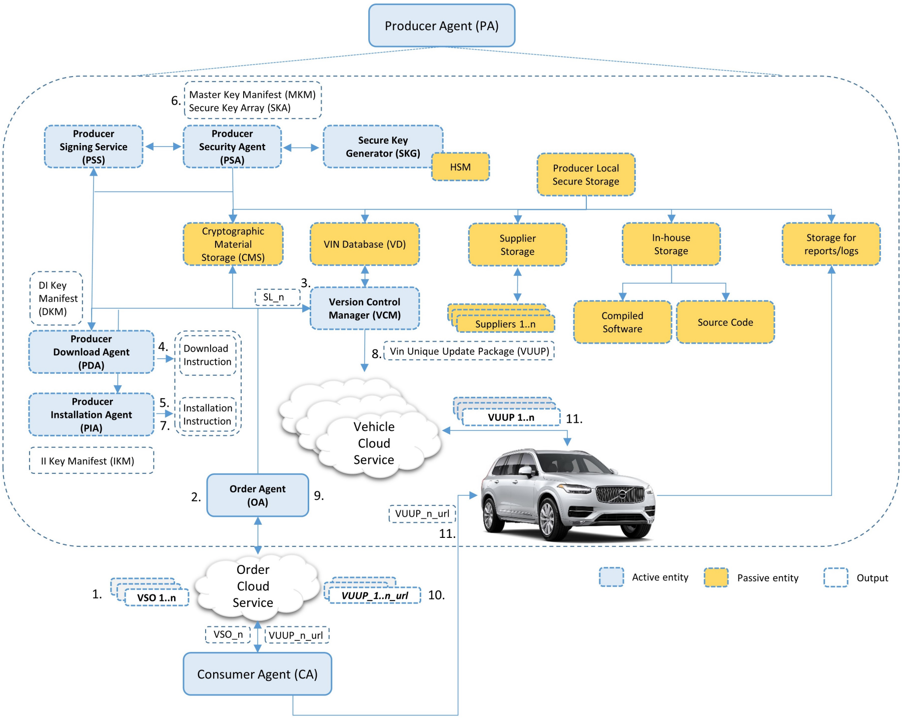


Figure 6.1: Data distribution in the backend

6.4 The Software Update Process

In this section, we dive into the details of the complete software update process in UniSUF. Explanations of the abbreviations used can be found in Table 6.1.

Table 6.1: Abbreviations

Abbreviations	
Vehicle Identification Number (VIN)	The VIN number is a vehicle unique fingerprint, and is composed of 17 characters.
Producer Agent (PA)	Parent entity consisting of many children entities covering backend requirements.
Producer Security Agent (PSA)	Responsible for handling cryptographic material in the backend systems.
Producer Signing Service (PSS)	Executes signing requests i.e., returns signatures of hash values requested by authenticated entities.
Order Agent (OA)	Responsible for managing software requests from consumers.
Secure Key Generator (SKG)	PSA uses this module for the secure generation of key material.
Secure Key Array (SKA)	An array that PSA creates with cryptographic material related to a VIN unique software package.
Version Control Manager (VCM)	Responsible for management of software versions related to unique vehicles and for repackaging of data into the final VUUP file.
Producer Download Agent (PDA)	Creates the instructions for the download of software for a certain VIN.
Producer Installation Agent (PIA)	Creates the diagnostic instructions for installation of software for a certain VIN, including retrieving necessary cryptographic material.
VIN Database (VD)	Stores VIN unique data related to software.
Cryptographic Material Storage (CMS)	Secure storage of cryptographic material.
Download Key Manifest (DKM)	The manifest that contains the DKM session key with the policy for decryption of the download instruction.
Installation Instruction Key Manifest (IKM)	Contains the IKM session key with policy for decryption of the installation instruction.
Master Key Manifest (MKM)	Contains MKM session keys with policies for decryption of cryptographic data.
Vehicle Unique Update Package (VUUP)	The update package that includes information to perform a complete vehicle software update, e.g., software download instructions, installation instructions and cryptographic material.
Consumer Agent (CA)	The parent entity to the children entities covering vehicle requirements for the software installation process. The localization for children entities can be adapted to accommodate various use cases, e.g., OTA, workshop, and factory.
Consumer Download Agent (CDA)	Executes download instructions and retrieves required software files to local storage.
Consumer Installation Agent (CIA)	A diagnostic client responsible for the execution of installation instructions and requests to CSA for the execution of cryptographic material.
Consumer Security Agent (CSA)	A trusted execution environment (TEE), with prestored certificates between vehicle manufacture and CSA; which enables secure transfer and execution of cryptographic material from the backend to the vehicle.
Key Wrapping (KW)	The process of encrypting one key with the use of another symmetric or asymmetric key, to securely store or transmit it over an untrusted channel.
Key Manifest (KM)	Used to define policies and relations for certain keys. Keys are secured with KW, where encrypted keys and policies are signed, giving rise to a KM.

6.4.1 Encapsulating Data into a VUUP file

Producer Agent (PA): data distribution in backend. Figures 6.1, 6.2 and 6.3 describe the process of creating a complete *VUUP* file. The 11 steps

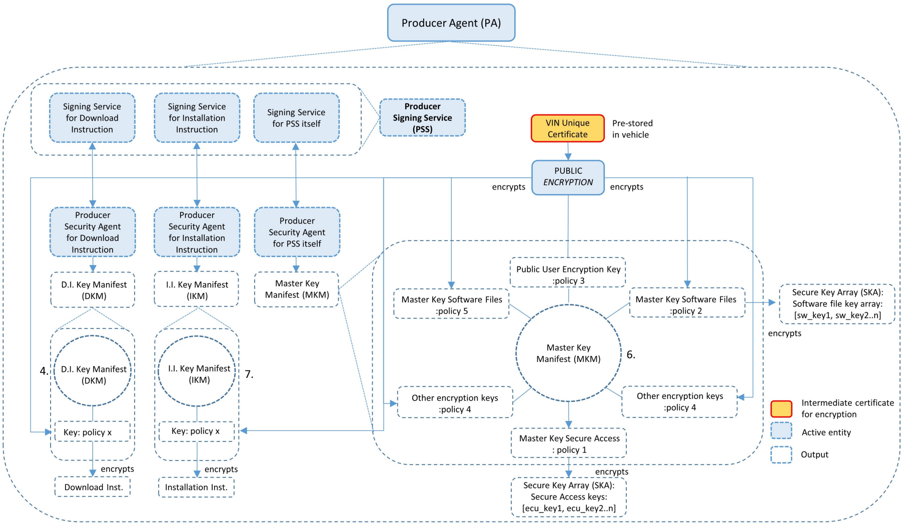

Producer Agent (PA)
Signing Service for Download Instruction
Signing Service for Installation Instruction
Signing Service for PSS itself
Producer Signing Service (PSS)
VIN Unique Certificate
Pre-stored in vehicle
PUBLIC ENCRYPTION
encrypts
encrypts
Producer Security Agent for Download Instruction
Producer Security Agent for Installation Instruction
Producer Security Agent for PSS itself
D.I. Key Manifest (DKM)
I.I. Key Manifest (IKM)
Master Key Manifest (MKM)
Public User Encryption Key :policy 3
Master Key Software Files :policy 5
Master Key Software Files :policy 2
Secure Key Array (SKA): Software file key array: [sw_key1, sw_key2..n]
encrypts
4.
D.I. Key Manifest (DKM)
I.I. Key Manifest (IKM)
7.
Master Key Manifest (MKM)
6.
Other encryption keys :policy 4
Other encryption keys :policy 4
Key: policy x
Key: policy x
Master Key Secure Access : policy 1
Intermediate certificate for encryption
Active entity
Output
encrypts
encrypts
encrypts
Download Inst.
Installation Inst.
Secure Key Array (SKA): Secure Access keys: [ecu_key1, ecu_key2..n]

Figure 6.2: Data distribution in the backend in relation to cryptographic material

described below are indicated by numbers where relevant in the Figures 6.1, 6.2 and 6.3.

1. Order request. The *Consumer Agent (CA)* in the vehicle, local workshop, or any other consumer, places a signed order on behalf of a *Vehicle Identification Number (VIN)* (i.e., a *Vehicle Signed Order (VSO)*). A *VSO* should contain a complete vehicle readout and be signed by the entity which creates the order. *VSOs* are placed in the *Order Cloud Service* queue. *Output: VSO_n.signed;*

2. Initiate VCM with VSO file. The *Order Agent (OA)* pulls *VSOs* from the *Order Cloud Service* queue, verifies the *CA* signature of the *VSO*, and requests initiation by *VCM* with this *VSO*.

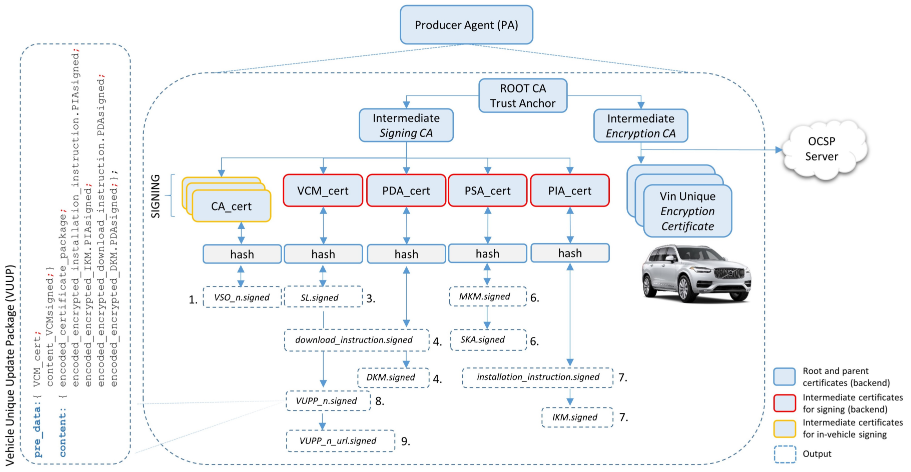
3. VCM creates an SL file with VIN unique software information. *VCM* receives a *VSO* from *OA* for a certain *VIN*. *VCM* validates the signature of the *VSO* and retrieves the latest available software versions and *VIN* vehicle data from the *VIN Database (VD)*. VIN data in *VD* is compared with actual vehicle software readout in the *VSO*. Software deviations are handled, and a signed *Software List (SL)* is created from information in the *VSO* and *VD* and is sent to *PDA*, *PIA*, and *PSA*. *Output: SL.signed;*

4. PDA creates download instructions. *PDA* verifies the *SL* and creates download instructions (list of software urls) for all ECUs based on the *SL*. *PDA* requests a DKM (*Download Instruction Key Manifest*) session key from *PSA* and encrypts the download instructions with this key. Next, this key is encrypted with a vehicle unique public certificate retrieved from *CMS*, where the certificate is validated for authenticity towards the *Root CA* and *OCSP* (Online Certificate Status Protocol). The encrypted DKM session key and a policy that dictates the association to the download instructions give rise to the *DKM*. A hash is calculated of the encrypted download instructions and the *DKM* separately, and signature requests are sent to *PSS* on behalf of *PDA*, which replies with two separate signatures. *Output: download_instruction.signed; DKM.signed; PDA_cert;*

5. PIA creates installation instructions. *PIA* verifies the signature of the *SL* and creates installation instructions for all ECUs based on the *SL*. *Output: installation_instruction;*
6. PSA requests cryptographic material. *PSA* verifies the *SL* and retrieves the required cryptographic material for software related to the received *SL* from *CMS*, such as keys for unlocking ECUs, privileged diagnostic requests, and software decryption keys. For each category of cryptographic material, *PSA* generates an MKM (*Master Key Manifest*) session key, where each key is associated with that specific category policy. MKM keys are in turn encrypted separately with a vehicle unique public certificate retrieved from *CMS* (same certificate as in Step 4), where the vehicle unique certificate from *CMS* is validated for authenticity towards *Root CA* and *OCSP*. The encrypted MKM keys with each respective category policy give rise to the *MKM*. A key array named *SKA* is created, which includes a sub-array for each category with separately encrypted key data, encrypted with the MKM key which belongs to that specific category. For example, *SKA* can include an array of encrypted symmetric keys used to encrypt/decrypt the relevant software update files, so called sw_keys (cf. Section 6.3.3), and an array of encrypted security access keys used for unlocking relevant ECUs. A hash is calculated of the *SKA* and *MKM*, where after signature requests are sent to *PSS* which replies with two separate signatures. *Output: MKM.signed; SKA.signed; PSA_cert;*
7. PIA retrieves the signed MKM and SKA from the PSA, and encrypts/signs the installation instruction. *PIA* request the signed *MKM* and the signed *SKA* from *PSA*. *MKM* and *SKA* signatures are validated where after *MKM* and *SKA* are included as part of the installation instructions. *PIA* requests an IKM (*Installation Instruction Key Manifest*) session key from *PSA* and encrypts the installation instructions with this key. The IKM session key is then encrypted with a vehicle unique public certificate retrieved from *CMS* (same certificate as in Step 4.), where the certificate is validated for authenticity towards the *Root CA* and *OCSP*. The encrypted IKM session key and a policy that dictates the association to the installation instructions give rise to the *IKM*. A hash is calculated of the encrypted installation instructions and the *IKM* separately, and signature requests are sent to *PSS* on behalf of *PIA*, which replies with two separate signatures.
Output: installation_instruction.signed; IKM.signed; PIA_cert;
8. VCM creates the VUUP file.
Input: download_instruction.signed; DKM.signed; installation_instruction.signed; IKM.signed; PDA_cert; PIA_cert;
VCM retrieves the generated data from *PDA* and *PIA*. Certificates are fetched from CMS and validated for authenticity towards the *Root CA* and *OCSP*, signatures are validated with the respective certificate and all the data is repackaged into *VUUP* content. A hash is calculated of the *VUUP* content, and a signature request is sent to *PSS* on behalf of *VCM*, which replies with a signature. The signed VUUP is uploaded to the *Vehicle Cloud Service* together with its VCM certificate. *Output: VUPP_n.signed; VCM_cert;*
9. VCM notifies OA. *VCM* notifies *OA*, that the order is ready and supplies a signed URL to the VUUP file. *Output: VUUP_1..n_url.signed;*
10. OA adds the url to the VIN unique VUUP in the Order Cloud Service. The *OA* validates the signature of url and thereafter adds the url in

the *Order Cloud Service*.

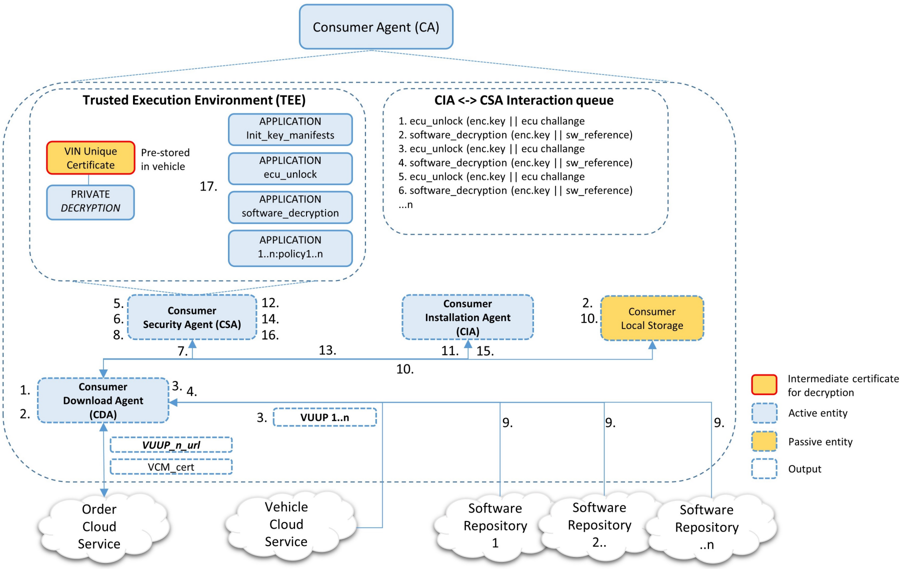
11. CDA requests status. The *CDA* pulls status from the *Order Cloud Service* (via the *CA*) to indicate that updates are available for download via the signed VUUP_n_url. If no updates yet are available, the signed url will be empty.



Vehicle Unique Update Package (VUUP)

```
pre_data: { VCM_cert;
            content_VCMsigned; }
content:  { encoded_certificate_package;
            encoded_encrypted_installation_instruction.PIAsigned;
            encoded_encrypted_IKM.PIAsigned;
            encoded_encrypted_download_instruction.PDAsigned;
            encoded_encrypted_DKM.PDAsigned; };
```

Figure 6.3: Data distribution in the backend in relation to signing



Consumer Agent (CA)
Trusted Execution Environment (TEE)
VIN Unique Certificate
Pre-stored in vehicle
PRIVATE DECRYPTION
APPLICATION Init_key_manifests
APPLICATION ecu_unlock
APPLICATION software_decryption
APPLICATION 1..n:policy1..n
17.
CIA <-> CSA Interaction queue
1. ecu_unlock (enc.key || ecu challange
2. software_decryption (enc.key || sw_reference)
3. ecu_unlock (enc.key || ecu challange
4. software_decryption (enc.key || sw_reference)
5. ecu_unlock (enc.key || ecu challange
6. software_decryption (enc.key || sw_reference)
...n
5. 6. 8.
Consumer Security Agent (CSA)
12. 14. 16.
Consumer Installation Agent (CIA)
2. 10.
Consumer Local Storage
7.
13.
11.
15.
10.
1. 2.
Consumer Download Agent (CDA)
3. 4.
3. VUUP 1..n
VUUP_n_url
VCM_cert
9.
9.
9.
Order Cloud Service
Vehicle Cloud Service
Software Repository 1
Software Repository 2..
Software Repository ..n
Intermediate certificate for decryption
Active entity
Passive entity
Output

Figure 6.4: Data distribution to the vehicle

6.4.2 Decapsulating the VUUP file

Consumer Agent (CA): data distribution to vehicle. For the *CA*, the process can be considered as the *PA* process reversed. The 17 steps described

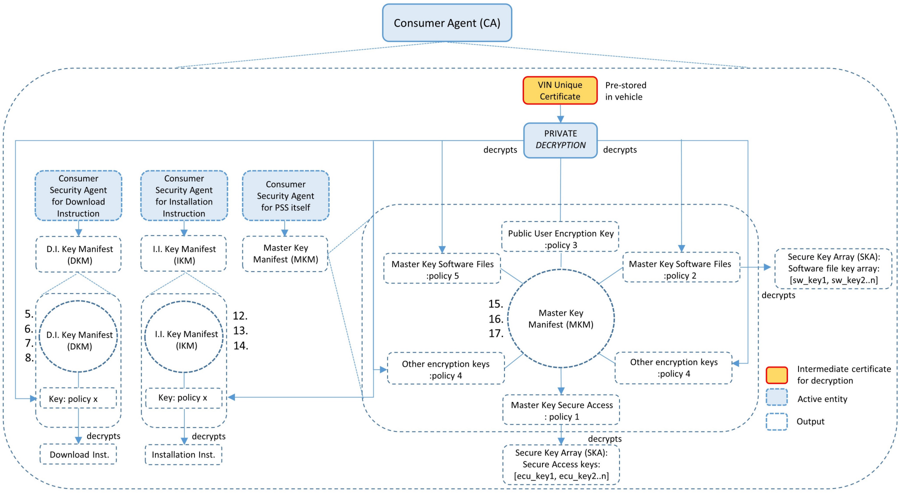

Consumer Agent (CA)
VIN Unique Certificate
Pre-stored in vehicle
PRIVATE DECRYPTION
decrypts
decrypts
Consumer Security Agent for Download Instruction
Consumer Security Agent for Installation Instruction
Consumer Security Agent for PSS itself
D.I. Key Manifest (DKM)
I.I. Key Manifest (IKM)
Master Key Manifest (MKM)
5.
6.
7.
8.
D.I. Key Manifest (DKM)
I.I. Key Manifest (IKM)
12.
13.
14.
Key: policy x
Key: policy x
decrypts
decrypts
Download Inst.
Installation Inst.
Public User Encryption Key :policy 3
Master Key Software Files :policy 5
Master Key Software Files :policy 2
Secure Key Array (SKA): Software file key array: [sw_key1, sw_key2..n]
decrypts
15.
16.
17.
Master Key Manifest (MKM)
Other encryption keys :policy 4
Other encryption keys :policy 4
Master Key Secure Access : policy 1
decrypts
Secure Key Array (SKA): Secure Access keys: [ecu_key1, ecu_key2..n]
Intermediate certificate for decryption
Active entity
Output

Figure 6.5: Data distribution to the vehicle in relation to cryptographic material

below are indicated by numbers where relevant in the Figures 6.4, 6.5 and 6.6.

1. The CDA requests software updates.
Input: VUUP_n_url.signed; VCM_cert;
If there are updates available, the CDA receives a signed VUUP_n_url and VCM_cert, where the certificate is validated for authenticity towards the *Root CA* and *OCSP*, where after the signature of VUUP_n_url is validated using the received VCM_cert.

2. Download of VUUP. Mutual authentication is performed towards the *Vehicle Cloud Service* and the signed VUUP_n is downloaded to *Consumer Local Storage*. *Output: VUPP_n.signed;*

3. Validate VUUP. The signature of VUUP_n is validated with VCM_cert, and VUUP_n is decapsulated to produce the signed contents of download instructions, DKM, installation instructions, IKM as well as the included PDA_cert and PIA_cert. *Output:*
download_instruction.signed; DKM.signed; installation_instruction.signed; IKM.signed; PDA_cert; PIA_cert;

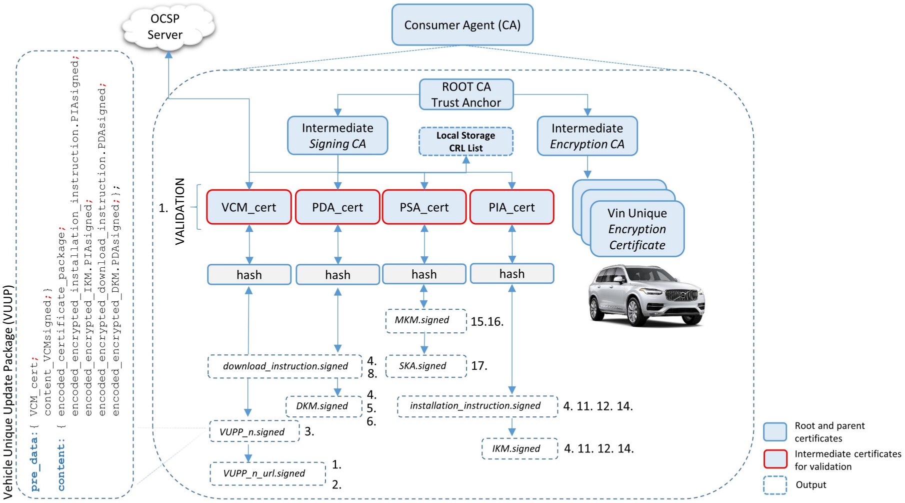
4. Validate data within VUPP. Certificates are fetched for online cases or retrieved from the VUUP file for offline cases. The signatures for the download instructions and the *DKM* are validated with the PDA_cert, and the signatures for the installation instructions and the *IKM* are validated with the PIA_cert.

5. DKM Key manifest initiation. The *CDA* requests the *CSA* to initialize the *DKM*, by providing the *DKM.signed* and PDA_cert.
Output: DKM.signed; PDA_cert;

6. CSA associates DKM with TEE application. The signature of *DKM* is validated with PDA_cert. The *DKM* session key is decrypted with the pre-stored vehicle unique private certificate and associated with the TEE application according to the policy in the *DKM* manifest, i.e., for decrypting download instructions.

7. Request decryption of download instruction. The *CDA* provides the signed download instructions to the *CSA* and requests decryption.

Output: download_instruction.signed;
8. Perform decryption of download instruction. The *CSA* validates signature of the download instruction with PDA_cert, decrypts with the DKM session key from the *DKM* in accordance with policy (i.e., decrypting download instructions) and returns the decrypted download instructions to the *CDA*.
Output: download_instruction;
9. Download of software files. The *CDA* performs mutual authentication towards various software repository sources and downloads encrypted signed software files to *Consumer Local Storage* using the download instructions. The *CDA* validates signatures of all encrypted software files with the VCM_cert.



```
pre_data: { VCM_cert;
            content_VCMsigned;}
content:  { encoded_certificate_package;
            encoded_encrypted_installation_instruction.PIAsigned;
            encoded_encrypted_IKM.PIAsigned;
            encoded_encrypted_download_instruction.PDAsigned;
            encoded_encrypted_DKM.PDAsigned;};
```

Figure 6.6: Data distribution to the vehicle in relation to validation

Consumer Agent (CA): data execution in vehicle. After data distribution to the vehicle has been completed, the following steps describe the installation of the software update through data execution in the vehicle. These steps can be performed completely offline.
10. Initiation of pre-state phase. The *CDA* requests to start installation of software by sending the signed installation instructions, signed IKM, and the PIA_cert to the *CIA*.
Output: installation_instruction.signed; IKM.signed; PIA_cert;
11. Reboot to secure state. The *CIA* validates the PIA_cert for authenticity towards *Root CA* and *OCSP*. The signatures of the installation instructions and *IKM* are then validated with the PIA_cert. *CIA* then reboots to an offline secure state; ready for pre-state installation processes. PIA_cert is validated again after reboot, against *Root CA* and an offline CRL list; and the signature of IKM is validated with PIA_cert, where after the *CIA* requests *IKM* initialization by sending the signed IKM and PIA_cert to the *CSA*. *Output: IKM.signed; PIA_cert;*
12. IKM key manifest initiation. The *CSA* validates the PIA_cert for authenticity towards *Root CA* and an offline *CRL*. The *CSA* then validates the *IKM* signature with PIA_cert, where after the IKM session key within *IKM* is decrypted with the pre-stored private asymmetric unique key and associated

according to policy in the *IKM*, i.e., to be used for decrypting installation instructions.
13. Request of decryption of installation instruction. The *CIA* provides the signed installation instructions to the *CSA* and requests decryption.
Output: installation_instruction.signed;
14. Decryption of installation instruction. The *CSA* validates the signature of the installation instructions with PIA_cert, decrypts with the IKM session key from the *IKM* in accordance with policy (i.e., decrypting installation instructions) and returns the decrypted installation instructions to the *CIA*.
Output: installation_instruction;
15. Request MKM key manifest initiation. The *CIA* retrieves the encapsulated signed *MKM* and *SKA*, and PSA_cert from the decrypted installation instructions. The *CIA* validates the PSA_cert for authenticity towards *Root CA* and an offline *CRL*, and verifies signatures of the *MKM* and the *SKA* with the PSA_cert. *CIA* then requests *MKM* initialization by sending the signed *MKM* and PSA_cert to the *CSA*. *Output: MKM.signed; PSA_cert;*
16. MKM key manifest initiation. The *CSA* validates the *MKM* signature with the PSA_cert. MKM session keys within the *MKM* are decrypted with the pre-stored private asymmetric unique key and are associated with applications according to policy in the *MKM*.
17. Secure CIA - CSA interface established. Peri-state. The *CIA - CSA* communication interface is now initialized. The *CIA* can request decryption of software from the *CSA* by sending the encrypted file (or path/link) together with the corresponding encrypted sw_key retrieved from the *SKA*. Before decryption can start, the *CSA* validates the authenticity of software files with the VCM_cert and aborts the decryption request from the *CIA* if this fails. On the other hand, if it is successful, the *CSA* then decrypts the encrypted sw_key with the MKM session key for software files from the *MKM* and uses the sw_key to decrypt the software file. This interface is also used for unlocking ECUs using, e.g., security access to authorize the update. In this case, the challenge from the ECU is sent to the *CSA* together with the corresponding encrypted security access key from the array in *SKA*. *CSA* decrypts the encrypted security access key and processes the challenge from the ECU and can provide the results to the *CIA*. This approach allows the *CIA* to perform ECU unlock without exposing the security access key outside of *CSA*. The *CIA* is after this step ready to stream out software to the ECU.

6.4.3 Post-State Activities

CSA needs to have the possibility to sign post-state installation data, such as installation reports and logs potentially affecting upcoming software updates. A unique session signing key can be transported via *MKM* to *CSA* which can handle signing requests within a trusted application isolated within the *TEE*. The corresponding validation key can be stored in *CMS*. Part of post-state is to perform a complete vehicle software version request (readout). To provide authenticity, the readout can be signed by supported ECUs (e.g., if they contain pre-stored private keys) and validated by *CSA* with the help of the corresponding validation keys attached to the *SKA*. These responses are then attached to the installation report. To provide confidentiality, *CSA* can encrypt installation reports and logs by using keys in *SKA*.

6.5 Implementation Considerations

The *CA* (all consumer entities) can as shown in Figure 6.4 be located in an ECU in the vehicle used for over-the-air updates or in a client workshop computer with a separated *CSA*. *CSA* in this case can be implemented in a hardware security device such as a Yubico key [269] or a smart card, with a pre-stored encryption/decryption certificate and a pre-stored *Root CA* acting as a trust anchor for validating certificates. Vehicle manufacturers can provide these hardware security devices to workshops, and also have full control to manage and revoke them. Depending on both security and performance requirements *CSA* can also be placed in a workshop HSM or even located in the cloud. Because of the proposed entity separation (implemented as modules), other approaches are also possible, such as integrating *CDA* and *CIA* in an update tool with *CSA* integrated in hardware or separated. It is also possible to use *CDA* separately (outside the vehicle) and securely push the update package to the in-vehicle *CIA* (e.g., via mutually authenticated communication). *CIA* then validates and executes the installation instructions and uses the ECU-internal *CSA* to perform secure transfer and execution of cryptographic operations. The different entities can be securely containerized out in the cloud or kept within vehicle manufacture premises. This solution fulfills the constraints and conditions stated in Section 6.2 and is therefore highly adaptable to accommodate various scenarios within the automotive industry.

6.6 Related Work

Samuel et al. suggest using a layered approach with the use of different roles and keys called *The Update Framework (TUF)* to ensure the integrity of the downloaded data, however, this approach does not consider the installation of these updates and is not adapted for vehicles [270]. In [166] T. Kuppusamy, propose an implementation and adaption of the TUF framework named *Uptane* for vehicle over-the-air updates, where the authors add more metadata to improve its resilience. Another approach proposed by Idress et al. [271] suggests deploying a new architecture where all in-vehicle ECUs use HSMs for over-the-air updates. In [272] Mahmud et al. propose an architecture that relies on sending multiple copies to secure the software update, an approach which we believe is not realistic due to infrastructure constraints. M. Steger et al. propose a framework named *SecUp* which uses handheld devices to wirelessly connect and update vehicles over an IEEE 802.11s mesh network for local environments (i.e. factory and workshop) [273]. In [274] Nilsson et al. present an approach for securing firmware updates over-the-air by combining encryption, hashing, and signing of firmware by chaining fragments. In [275] Nilsson et al. continue their work on hash-chain verification and suggest an over-the-air update framework that validates firmware after it is flashed to the ECUs, however, this requires all in-vehicle ECUs to be adapted to this approach and that integrity verification of the download is solved by other means.

However, the aforementioned solutions lack necessary details for a *unified* and *versatile* approach that supports updates over-the-air; from a workshop computer; at the factory production; use of diagnostic update tools; and third-party vehicle platform users e.g., using the vehicle as a base controlled by other autonomous systems. As a case in point, *Uber* is using the *Volvo Cars* platform in their fleet of cars [267]; a scenario which most likely will become more prevalent in the future due to increased sustainability requirements. Thus, solutions such as ride-sharing will probably be more common where collective fleets of cars require integrating 3rd party hardware and software which are dependent on a unified software update framework. UniSUF supports 3rd party components by appending its related data to the VUUP file i.e., adding additional instructions and at the same time keeping the VUUP file intact. Moreover, other details are missing in current solutions such as required installation instructions including handling of necessary pre-, peri- and post-state phases and secure transport and secure execution of ECU-specific cryptographic keys. UniSUF considers all these three states, and ensures a secure transport to a trusted execution environment, following a secure execution for all sensitive data. Many existing solutions also consider changes to all ECUs which usually is not possible; something which is not required by UniSUF. The mentioned versatility of UniSUF can keep required adaptions of the vehicle as well as the required cost to a minimum.

6.7 Future Work and Conclusion

We have contributed with a comprehensive and novel unified software update framework named UniSUF, well aligned with industry stakeholders. As part of future work, we have already begun defining an attacker model and started a security analysis of our proposed solution. We aim to perform a more detailed evaluation of UniSUF, including a discussion on the fulfilled requirements as well as a comparison to other approaches, in a future paper.

UniSUF is made to accommodate various scenarios for the automotive domain by encapsulating needed data into one single file, a Vehicle Unique Update Package (VUUP). This vehicle unique file can be processed within a vehicle ECU, using a workshop computer, at factory production, with a diagnostic update tool, or in other compositions. Moreover, the complete update process can be performed without any external communication dependencies, since all files are inherently secured. A continuous secure software update process is a prerequisite for facilitating vehicle resilience towards cyber attacks in a rapidly changing environment. We believe our contributions in this paper can facilitate further research in this area, towards securing the connected car.

Chapter 7

Secure Vehicle Software Updates: Requirements for a Reference Architecture

Format-adapted version that appeared in IEEE Vehicular Technology Conference 2023

K. Strandberg, U. Arnljung, T. Olovsson, D. K. Oka

Abstract. A modern vehicle is no longer merely a transportation vessel. It has become a complex cyber-physical system containing over 100M lines of software code controlling various functionalities such as safety-critical steering, brake, and engine control. The amount of code is anticipated to rise to around 300M lines of code by 2030. Furthermore, even well-tested code will contain more than one bug per 1000 lines of code. Thus, it can be expected that there will be around 100k bugs in a modern vehicle and around 300k bugs in a few years, where some might have a safety-critical impact. Automotive companies are transforming into software companies with more software developed in-house. The ability to hastily and securely patch vulnerabilities has become vital and is a prerequisite when securing modern cars. The UN Regulation No. 156 and the ISO 24089 emphasize the ability to update vehicle software securely.

Consequently, we focus on securing the vehicle software update process. Our contributions include defining an attacker model and general security requirements. We further map these requirements to common security goals and directives to ensure broad coverage. Additionally, we present UniSUF, a secure and versatile approach to vehicle software updates. We identify entities involved during vehicle software updates, perform a threat assessment, and map the identified threats to security goals and requirements. The results highlight a secure framework with high industrial relevance that can be used as a reference architecture to guide securing similar software update systems within automotive and related areas such as cyber-physical systems, internet-of-things, and smart cities.

7.1 Introduction

The complexity of software within the automotive domain is increasing at a high pace, and as a result, the number of potential software bugs increases as well. Hence, software updates mitigating vulnerabilities need to be applied regularly. The automotive industry has requirements for numerous software deployment scenarios, such as over-the-air, in workshops, and in factories, with or without Internet access [18]. However, there is a risk that if the software update process is vulnerable, it might leverage the potential for malicious code reaching in-vehicle systems causing life-threatening hazards such as manipulated brakes, steering, and engine control. Therefore, UN Regulation No. 156 [15] and ISO 24089 [11] provide demands on secure vehicle software updates.

A vehicle is a distributed system with dependability and real-time requirements and can contain over 150 computers running various operative systems, further interconnected using many different communication buses and protocols. Thus, performing secure software updates to vehicles requires a rather complex approach.

The software update process can be divided into *data distribution* and *data execution*, where the former concerns the difficulties of securely distributing all needed data, such as software files, installation instructions (diagnostic commands), and cryptographic material, to the entity responsible for the installation process for all in-vehicle computers, i.e., Electronic Control Units (ECUs). The latter can be divided into pre-, peri- and post-state for the installation process. Pre-state refers to the preparation necessary to execute before the installation process can start, including a version control validation for all ECUs by comparing vehicle-unique software versions in a database with the actual vehicle and handling deviations. Additionally, it may be necessary to disable firewalls and Intrusion Detection/Prevention Systems to enable unlocking and placing ECUs in programming mode and verify that the vehicle is in a state that allows software updates, e.g., parking mode with a secure offline state.

The peri-state involves potential validation, decryption, and installing and transferring the new software to affected ECUs. Post-state is to perform the necessary validation of the update, such as securely creating installation reports and logs that may affect upcoming installation processes. For example, the installation report can include information about corrupt in-vehicle data, such as invalid cryptographic keys or faulty ECUs, which must be solved by downloading additional updates or replacing hardware. Alternatively, the installation report can serve as proof of a successful update.

Furthermore, every vehicle is unique and needs to have unique *data distribution* and *data execution.* Thus, a unique vehicle configuration, multiple software files for every ECU, many unique cryptographic keys, and ECU-specific diagnostic requests are required. For instance, special cryptographic keys to turn off security functionality that might otherwise block the installation process.

The ECU installation process typically uses Unified Diagnostics Services (UDS), a communication protocol specified in the ISO 14229-1 standard [21]. UDS is also used for various tasks such as reading out fault codes, activating or deactivating firewalls, changing operations mode (e.g., driving or passive), and testing functionality. Additionally, there are different security levels related to diagnostics. To execute at a particular level, the entity responsible for the installation needs access to the corresponding key to unlock this level. These keys need to be securely distributed and used securely in the execution. Securing the distribution, storage, pre-, peri-, and post-execution processes w.r.t. the infrastructure and all in-vehicle processes is challenging.

Contributions. We have identified general requirements to ensure a secure software update process. These requirements fulfill common security goals for cyber-secure vehicles. Moreover, we present a reference architecture named UniSUF based on previous work [18]. We validate the usability and security of our reference architecture by identifying an attacker model and performing a threat assessment. Finally, we identify mitigation mechanisms and map the specific threats to security goals and requirements to strengthen the robustness and design of UniSUF for a broad industry adaption with UN Regulation No. 156 in mind.

7.2 Attacker Model

There can be various threat actors, such as cyber terrorists, foreign countries, hacktivists, and vehicle owners [276]. However, we assume a common agenda where someone aims to manipulate the software update process or the software itself at any entity or during communication between entities throughout the software update process. For instance, the intent can be to recover and exploit secret signing or obtain encryption keys used during the software update process. The latter might enable disabling firewalls or switching ECUs into programming mode to enable update capabilities. Additionally, attackers might want to decrypt software files to reverse engineer and gain insight into its contents affecting the intellectual property and try to find vulnerabilities, e.g., through analysis of safety-critical systems. Thus, the attacker's ultimate goal is to exploit the software update system so that malicious or unauthorized software providing additional or altered functionality reaches the in-vehicle system, for instance, to gain and maintain remote persistence.

7.3 Methodology

Related work by us on the resilience of vehicles against security threats called Resilient Shield [276] provides a list of common security goals (SG) and directives (D), shown in Table 7.2, developed from an analysis of cyber attacks on vehicles over a ten-year timespan to create a common baseline for a cyber-secure vehicle. We have identified general security requirements for software updates summarized in Table 7.1 and mapped them to the SG and D from Resilient Shield as security claims in Table 7.2. We further use Goal Structuring Notation (GSN) [277] to present proofs for claims in a graphical manner to map these claims (i.e., SG and D) to the general requirements in Table 7.1, as illustrated in Figure 7.1. Additionally, we map threats and elaborate on the fulfillment of requirements in Section 7.4.2. Thus, we achieve broad coverage by ensuring requirements covering established security goals and enhanced security by fulfilling these requirements. For instance, as shown in Figure 7.1 and Table 7.2, SG1 is fulfilled by requirements R1 and R11 and reinforced by

Table 7.1: General Requirements

Requirement R1: infrastructure and communication. The infrastructure, cryptographic algorithms, and key material shall follow best security practices. For instance, communication between backend entities shall encrypt communication and use proper authentication between entities. The same requirements shall be considered for in-vehicle entities directly related to the update framework.

Requirement R2: code review, testing and validation. When possible, external and internal code reviews shall be performed to detect vulnerabilities and deviations. Follow secure programming guidelines like the Secure Software Development Lifecycle (SSDLC). Continuous testing and validation shall be performed, such as positive/negative, vulnerability, fuzzing, penetration testing, and validation of, e.g., security controls.

Requirement R3: secure storage. Key management, such as generation and storage, shall be protected according to a high-security level within HSMs. Access to such should be highly restricted and only accessible to authorized users.

Requirement R4: redundancy. Relevant redundancy shall be used to switch to redundant entities during failures and compromises of entities or processes that are part of the update framework. For instance, traffic shall be redirected to redundant systems if appropriate during a denial of service attack.

Requirement R5: least privilege. Each backend entity shall be implemented according to the least privilege principle. i.e., each entity is responsible for securing its data (output), whereas data from other entities are validated and visible only on a need-to-know basis (input).

Requirement R6: separation of duties. The separation of duties shall be considered within the software update framework, where many separate entities shall be required to complete the distribution and execution processes.

Requirement R7: state awareness. Employed mechanisms and functions need to be robust against anomalies. The system shall be aware of its state and shall be able to switch to other states when anomalies are detected. For instance, when a cyber attack is detected, and if appropriate, the software updates system shall be able to abort, roll back and perform a retry from redundant entities.

Requirement R8: secure boot. State-of-the-art secure boot protection mechanisms shall be used where applicable, e.g., the installation responsible entity (the diagnostic client), including TEE environments. For example, the first image in a Chain of Trust has a ReadOnly Memory (ROM) and contains an immutable Hardware Trust Anchor (HTA), i.e., a Root of Trust code, including a root certificate. Hence, this image can be used to verify keys and signatures for upcoming loaded images such as TEEs.

Requirement R9: Intrusion Detection/Prevention Systems. IDSs/IPSs shall be used to detect and react to anomalies from normal communication patterns and known attacks.

Requirement R10: fault-tracing and forensics. Events related to software installation and security events (e.g., turning off the firewall) shall be securely stored to enable fault tracing and forensics investigations. For instance, traceability regarding time and source for events shall be possible.

Requirement R11: secure algorithms. The type and lifetime of key material concerning its context, future maintainability, and implementation shall be considered, e.g., using quantum-resistant algorithms and a set validity time for key material. However, the author intends not to recommend the type of algorithms to use.

Requirement R12: authenticity software. All encrypted software files shall be validated for authenticity before decryption by the installation responsible entity. Furthermore, supported end nodes shall perform another validation for decrypted software files. These two separate validation steps shall be based on different signing keys. The intermediate certificates shall be fetched, received, or pre-stored and always validated via OCSP or CRL requests and against a specific root certificate before being used.

Requirement R13: secure storage, freshness and authenticity within TEE. a) The private vehicle unique decryption key at the consumer (receiver) side shall be secured according to best security practices and only accessible inside the TEE. Furthermore, the public counterpart, i.e., the encryption key, shall be validated for authenticity before use towards Root CA, expiration date and revocation. b) The public key in R13a shall only be used to encrypt session keys. The validity time of these session keys shall be connected to the validity time of an update package. c) The input data (function calls) to TEE shall be validated for authenticity.

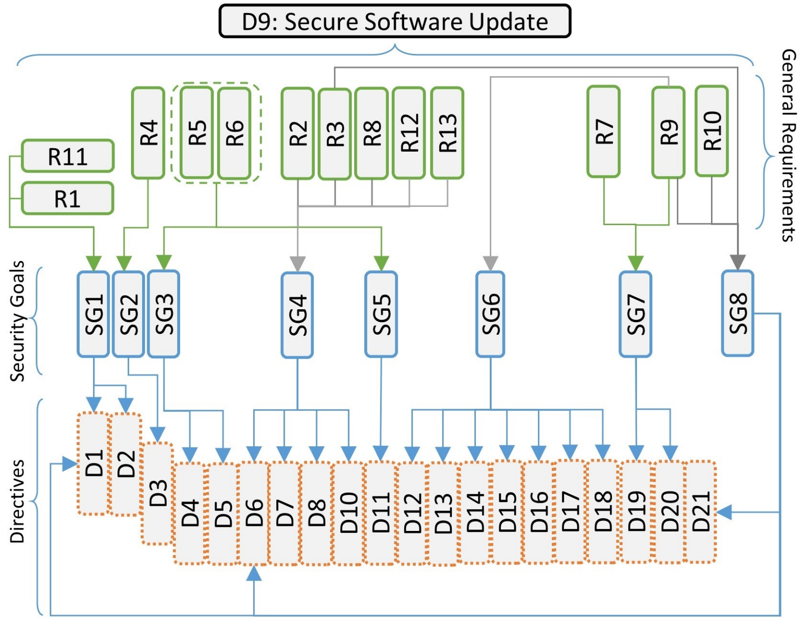


Figure 7.1: A Goal Structuring Notation over Secure Vehicle Software Updates

implementing the detailed directives D1 and D2, mitigating threats P1-P6, R, and C1-C3.

We further establish that the following security properties and principles are fulfilled by enforcing the following general requirements (cf. Table 7.1): *Confidentiality* by encryption and authorization [R1,R3,R5,R13]. *Integrity* and *authenticity* with hashes and signing [R1,R8,R12,R13]. *Authorization* and *isolation* between entities and their data by signing, containerization, virtualization, and trusted execution environment (TEEs) [R1,R3,R5,R13]. *Freshness* by a set validity time for the update package and associated session keys [R11-R13]. *Availability* and *reliability* using redundancy in the update system and support for implementing updates to the update framework when requirements change [R1-R13]. Principles such as the *least privilege* and *separation of duties* for entities are ensured on a need-to-know basis, and many separate entities are needed to complete vehicle updates [R5,R6]. Forensic capabilities with traceability by providing secure storage of events in logs [R10].

7.4 A Reference Architecture for Secure Vehicle Software Updates

This section introduces a reference architecture named UniSUF [18] that fulfills the general requirements presented in Section 7.3. We describe the reference architecture components, provide a threat assessment and elaborate on possible mitigations. UniSUF is made to cover the whole software deployment process, starting with securing the software update files to the installation process and finally creating post-process installation reports. UniSUF covers online

Table 7.2: Mapping of Requirements to Security Goals, Directives and Threats

[Requirement]	**[Security Goal]**	**[Directive]**	**[Threat]**
[R1,R11]	[SG1: Secure Communication]	[D1,D2]	[P1-P6, R, C1-C3]
[R4]	[SG2: Readiness]	[D3]	[P1-P6, R]
[R5,R6]	[SG3: Separation of Duties]	[D4, D5]	[P1-P6, R, C1-C3]
[R2,R3,R8,R12,R13]	[SG4:Secure Software Techniques]	[D6-D8,D10]	[P1-P6,R,C1-C3]
[R5,R6]	[SG5: Separation/Segmentation]	[D11]	[P1-P6, R, C1-C3]
[R9]	[SG6: Attack Detection and Mitigation]	[D12-D18]	[P1-P6, R, C1-C3]
[R7,R9]	[SG7: State Awareness]	[D19,D20]	[P1-P6, R, C1-C3]
[R3,R9,R10]	[SG8: Forensics]	[D1,D6,D21]	[P1-P6, R, C1-C3]

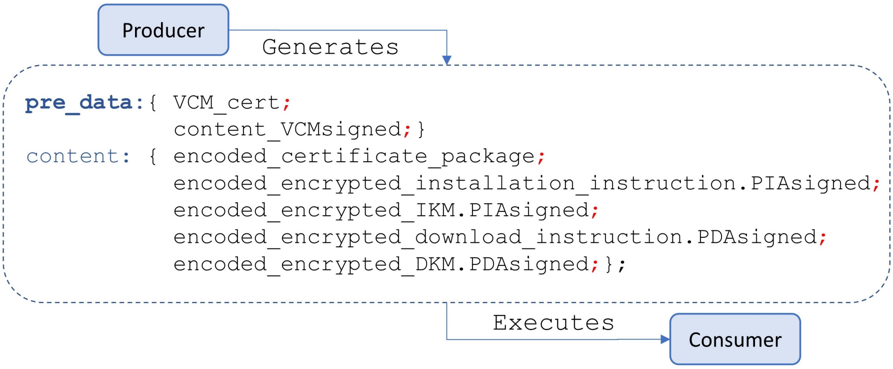
Directives

D1:Authentication, D2:Encryption, D3:Redundancy/Diversity, D4:Access Control, D5:Runtime Enforcement, D6:Secure Storage, D7:Secure Boot, D8:Secure Programming, D10:Verification & Validation, D11:Separation, D12:Specification/ Anomaly-based Detection, D13:Prediction of Faults/Attacks, D14:Adaptive Response, D15:Reconfiguration, D16:Migration/Relocation, D17:Checkpoint & Rollback, D18:Rollforward Actions, D19:Self-X, D20:Robustness, D21:Forensics

and offline updates without dependencies on the data distribution model or the software update storage location. Additionally, it supports updating of 3rd party components, which is an increasingly important requirement. For instance, more companies are using vehicle platforms extended with 3rd party components and need to update such components ideally using the same software update framework.

Three primary entities are involved in the update process: the producer, the consumer, and the repository. The first entity produces the software, which includes the automaker and 3rd party suppliers. The second consumes or uses the software and comprises the vehicle and its users. The third entity is a storage for the software before installation, such as various cloud sources and local storage points, e.g., local network workshop drives.

In UniSUF, all data required for a unique vehicle software update is encapsulated using encryption and signatures into layers of data producing one single update file, the Vehicle Unique Update Package (VUUP). As shown in Figure 7.2, VUUP contains *pre_data* and *content*. The first part, i.e., pre-data, contains a validation certificate and a signed hash of the rest of the content of the VUUP file. The actual content includes a package containing the required certificates to validate all signatures in the VUUP file. Moreover, the content includes installation and download instructions and all the required cryptographic keys. However, due to the complexity and cost impact, modifying the E/E to all ECUs to accommodate a new specific software update framework is rarely an option. Instead, UniSUF allows securing the software update processes without adding new functions to all in-vehicle ECUs. Potential security mechanisms in use, such as secure boot and software signing, in individual ECUs remain intact. UniSUF uses isolation techniques such as containers, virtual machines, or combinations of such for producer entities. Depending on the context, producer entities shall be implemented as isolated modules with controlled and secure communication, secured on-premises or in the cloud. Thus, the communication shall be encrypted and authenticated. Each entity



```
pre_data:{ VCM_cert;
           content_VCMsigned;}
content: { encoded_certificate_package;
           encoded_encrypted_installation_instruction.PIAsigned;
           encoded_encrypted_IKM.PIAsigned;
           encoded_encrypted_download_instruction.PDAsigned;
           encoded_encrypted_DKM.PDAsigned;};
```

Figure 7.2: The Vehicle Unique Update Package (adapted from [18])

is responsible for its data security, i.e., data is encrypted and signed at the source. Consequently, producer entities can validate the data. Still, only the intended consumer entity can read the information on a need-to-know basis via interaction with trusted applications running processes securely in isolation within a TEE on the consumer side.

7.4.1 Key management

UniSUF uses multiple signatures where producer entities have their specific signing keys. Entities are prevented from reading sensitive data created by other entities. Session keys are secured by encapsulation (i.e., asymmetric key wrapping) into layers within a final single file and transferred to the consumer. Encapsulation into layers implies that producer entities encrypt and sign their data (at the source). When data is retrieved, it is validated by the receiving producer entities that append their own encrypted and signed data. Finally, all information is collected and appended into a single VUUP file (cf. Figure 7.2 and 7.3). On the consumer side, the VUUP file is validated in its entirety, and further decapsulated, where after each internal component is validated in itself with its corresponding certificate. Policy-based keys are directed and bound to trusted applications by installation processes (pre-state) and isolated within the TEE. Thus, policies dictate how specific trusted applications executing in a trusted execution environment are allowed to use these keys. Decryption processes and keys are therefore isolated within these trusted applications and can be used for, e.g., decryption of software files, unlocking ECUs for software updates, and signing installation reports and logs.

7.4.2 Threat Assessment

This section lists the potential threats to UniSUF. The abbreviations used are listed in Table 7.3. We look at threat probability and give examples of mitigating these threats. Security goals, directives, and requirements are considered for each threat, according to Table 7.2. For instance, for threat R, SG4 and SG5 are most relevant, where R2 and R5 are mentioned as examples for mitigation. Although most requirements are applicable for all threats, there are some distinctions, e.g., R4 is typical for the producer and R13 for

Table 7.3: Abbreviations

Producer Security Agent (PSA) handles cryptographic material in the backend systems.

Secure Key Generator (SKG) is used by PSA to secure key material generation.

Vehicle Identification Number (VIN) is a vehicle unique fingerprint, and is composed of 17 characters.

Version Control Manager (VCM) manages software versions related to unique vehicles and creates the software list and repackage data into the VUUP file.

Producer Download Agent (PDA) creates the instructions for downloading software for a specific VIN.

Producer Installation Agent (PIA) makes the diagnostic instructions for software installation for a certain VIN, including retrieving necessary cryptographic material.

VIN Database (VD) stores necessary VIN unique data related to software.

Cryptographic Material Storage (CMS) is a secure storage of cryptographic material.

Consumer Download Agent (CDA) executes download instructions and retrieves all necessary software files to local storage.

Consumer Installation Agent (CIA) is a diagnostic client responsible for running installation instructions and requesting the execution of needed cryptographic algorithms.

Consumer Security Agent (CSA) has a trusted execution environment (TEE) with pre-stored certificates between vehicle manufacturers and the CSA, enabling secure transfers and execution of cryptographic material.

the consumer. We start with the producer, i.e., the backend side, then the repository, and end with the consumer, i.e., the vehicle side. We start with the perspective of isolated compromise, one entity at a time. Section 7.4.3, looks at a few examples of the consequences when the attacker has gained complete control over more than one isolated entity. Note that all attack vectors depend on the localization and implementation details. The probability is graded on a scale from low, medium and high and is based on the required time, expertise, tools, and proximity for the potential attack.

7.4.2.1 The Producer

We start with the perspective of isolated compromise on the backend side as visualized in Figure 7.3.

Threat P1: Producer Download Agent (PDA). *Location:* Within the automaker's internal infrastructure (on-premises) or via 3rd party suppliers, e.g., the cloud. *Functionality:* PDA receives a signed Software List (SL), validates the signature, and creates download instructions (URLs to software files). *Assumption:* PDA is compromised and under the complete control of the attacker. The attacker can manipulate the SL with instructions to download malicious data from an external source. Furthermore, the attacker can encrypt and sign the malicious download instructions with the correct keys. *Probability:* If PDA is secured according to the stated requirements, the probability of compromise is low. If located on-prem, it will likely be insider actors, and if located in the cloud, external attacks will likely be detected by intrusion detection systems. *Consequence and Mitigation:* Manipulated

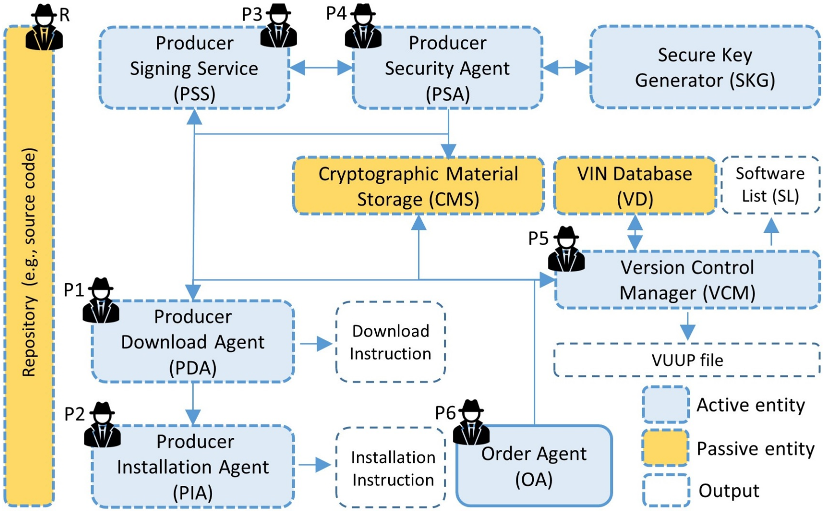

R
Repository (e.g., source code)
P3
P4
Producer Signing Service (PSS)
Producer Security Agent (PSA)
Secure Key Generator (SKG)
Cryptographic Material Storage (CMS)
VIN Database (VD)
Software List (SL)
P5
Version Control Manager (VCM)
P1
Producer Download Agent (PDA)
Download Instruction
VUUP file
P2
Producer Installation Agent (PIA)
Installation Instruction
P6
Order Agent (OA)
Active entity
Passive entity
Output

Figure 7.3: Overview of the Producer and Repository threats

download instructions can be included in the VUUP file. Still, since all software files shall be encrypted and signed with keys not accessible to the PDA, the update process will abort and disrupt the software update process. A mismatch between the installation instructions and available software files will be detected. Redundant system shall be available to take over, e.g., when an attack is detected, switch to another server [R4].

Threat P2: Producer Installation Agent (PIA). *Location:* Same as in *Threat P1*. *Functionality:* PIA receives the SL, validates its authenticity, and creates installation instructions based on SL. PIA also receives all required cryptographic keys in an encrypted form. Thus, not readable to PIA. *Assumption:* The attacker controls the PIA and can manipulate the diagnostic instructions. For instance, decide what parts of the software files should be installed, creating inconsistencies in the vehicle software. Possibly, the attacker intends to keep known exploitable vulnerabilities from being updated. *Probability:* Same as in *Threat P1*. *Consequence and Mitigation:* The consequence can be severe, depending on the existing type of vulnerabilities. The results of inconsistencies in vehicle software are difficult to anticipate. Post-installation processes will detect inconsistencies via a complete vehicle software readout, and additional updates can be issued using a redundant, non-compromised system [R4]. A warning message shall be given about a potential malfunction and compromised PIA. The signed SL file can be included in the VUUP file, validated, and used for a consistency check before performing installation processes on the consumer side. The SL file is composed and signed by another entity, i.e., the VCM [R6].

Threat P3: Producer Signing Server (PSS). *Location:* Within automaker premises if possible. *Functionality:* PSS consists of three sub-entities isolated to the PDA, PIA, and the PSA. After a mutual authorization process, PDA, PIA, or PSA request signatures of hash values from PSS. *Assumption:* The complete PSS is under control by an attacker who can freely decline or accept signing requests and return invalid or valid signatures. *Probability:*

Same as in *Threat P1 and P2. Consequence and Mitigation:* A disruption of the software update can occur. Invalid or non-existing signatures will abort the update process. If signing keys are compromised, these need to be revoked [R12]. Anomalies can be detected, and signing requests redirected to redundant systems [R4,R9].

Threat P4: Producer Security Agent (PSA). *Location:* Within automaker premises if possible. *Functionality:* PSA has access to the CMS and the SKG, where PSA retrieves and generates the required keys for the actual vehicle software installation process. *Assumption:* Secret keys are compromised and the threat actor has complete control over PSA, CMS, and SKG. *Probability:* Same as in *Threat P1-P3. Consequence and Mitigation:* The threat actor can disrupt the software update process by blocking key transferals or communicating faulty keys. IDS/IPS shall detect anomalies and react, e.g., switch to a redundant entity [R4,R9]. However, suppose the threat actor has physical access via the OBD-II port, debug ports, or connected directly to the communication bus. In that case, keys can be used to gain extended diagnostic privileges, for instance, turning off a firewall. We must then rely on end-nod security [R8, R12].

Threat P5: Version Control Manager (VCM). *Location:* Same as in *Threat P1-P4. Functionality:* VCM verifies the update request containing a fully signed software version readout from the vehicle, i.e., the current software composition for a unique vehicle. VCM retrieves the latest available software versions from the VIN database and creates a signed SL, which PDA, PIA, and PSA further process. Moreover, VCM validates input data from PDA and PIA and repackages the data into the VUUP file. *Assumption:* A threat actor can create an SL not corresponding with an approved combination of software versions and add faulty data into the VUUP file. *Probability:* Same as in *Threat P1-P4. Consequence and Mitigation:* Considering the amount of ECUs running various operating systems, aligned with the extensive amount of code, different software versions between ECUs might not be compatible and can cause unpredictable behavior. Thus, extensive testing is performed within the automotive industry towards different baselines, i.e., combinations of software versions [R2,R7].

A threat actor can prevent a specific software version with known vulnerabilities from being updated by stating that the ECU already has the correct software version installed. However, it will be detected by post-installation processes since a complete vehicle readout will be included in the logs and can thus be solved by issuing an additional update. Moreover, since VCM only repackage already signed data, any manipulation of such data will be detected [R5,R6]. If the issue remains, a redundant system shall be used, and a warning message of a potentially faulty VCM will be issued [R4]. Additionally, a consistency check towards an approved baseline can be performed by PDA, PIA, and PSA before the installation process and give a warning message when deviations between SL and an approved baseline are detected, and further allow or block the update depending on the detected deviations [R5,R6].

Threat P6: Order Agent (OA). *Location:* Same as in *Threat P1-P5. Functionality:* OA is responsible for managing the queue of vehicle software update requests. Verify that the update requests are authentic and initiate the update process via the VCM. *Assumption:* A threat actor has control over OA.

Probability: Same as in *Threat P1-P5*. *Consequence and Mitigation:* Threat actors can block or allow update requests. Blocking updates can be performed by claiming that the signature validation of software update requests failed or supplying a malicious URL to a faulty VUPP file. In the first case, no VUUP file will be created, and in the second case, the VUUP file will be made. However, the signature validation of an incorrect VUUP file will fail, and the update will be aborted [R5,R6]. A warning message shall be issued when there are signs of update failures, whereafter, a redundant system can be used [R4].

7.4.2.2 The Repository

As shown in Figure 7.3, a passive entity that contains, e.g., source code.

Threat R: Repository, supply chain and insider threats. *Location:* 3rd party suppliers and within automakers premises. *Functionality:* Software controls various parts of the vehicle, including safety-critical systems. *Assumption:* Attacks/threats on the supplier and internal side, e.g., manipulation of source code and attacks on servers. *Probability:* Low for automakers' own developed software and medium for 3rd party suppliers due to the complexity of the supply chain and the limited potential for code reviews. *Consequence and Mitigation:* Consequences can be severe, depending on the code. For instance, code might affect safety-critical ECUs directly or indirectly via other ECUs not being satisfactorily isolated. Automakers and external suppliers shall limit access to codebase [R5] and perform software code reviews [R2]. Code validation processes shall be established through the supply chain and internal automaker processes.

7.4.2.3 The Consumer

As shown in Figure 7.4, the focus of this section is on the threats concerning an individual vehicle. The threats are still dependent on implementation details, such as the location of the various modules [18]. For instance, the diagnostic client, i.e., CIA, can be part of an ECU in the vehicle but also executed externally via a diagnostic update tool connected to the OBD-II port.

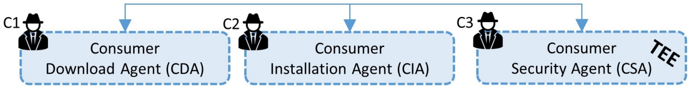


Figure 7.4: Overview of the Consumer Threats

Threat C1: Consumer Download Agent (CDA). *Location:* In vehicle or within an external tool. *Functionality:* CDA checks if updates are available and receive a signed URL to the unique VUUP file if that is the case. CDA then validates the URL's authenticity and downloads the VUUP file to local storage. Further, the VUUP file is validated and decapsulated. Whereafter all internal contents are validated with their respective certificate. The next step is for CDA to request the initiation of keys by the CSA. If successful, CDA can request a decryption process of the download instructions from CSA. *Assumption:* Threat actor has complete control of the CDA and can block the update process and manipulate the download URL to retrieve additional malicious files to local storage. *Probability:* We consider a medium probability for external tools and a low probability for in-vehicle implementations.

Consequence and Mitigation: From a CDA perspective, malicious data can only be downloaded, not executed. The attacker can allow CDA to finalize all steps with accurate data. In that case, still, the CIA will never execute any data from local storage without validating the signatures of the data. Secure boot protection mechanisms shall detect manipulation of CDA functionality [R8]. CDA and CIA can be integrated into the same ECU and be part of the same secure boot mechanisms. However, CDA can also be integrated into an external download tool, whereafter data is pushed to local vehicle storage, e.g., via mutual SSH [R1]. If the external CDA has been compromised, malicious data might reach local storage. Still, the CIA will not execute any data without the successful validation of signatures of the data [R6,R12]. Detected comprise of tools shall lead to revocation. Thus, mutual authentication shall fail for these devices concerning downloads from external sources and pushing data to the vehicle [R1,R11,R13].

Threat C2: Consumer Installation Agent (CIA).

Location: Same as C1. *Functionality:* The responsibility of the CIA is to execute installation instructions, thus installing decrypted and validated software files. Moreover, after the CIA successfully established a communication interface with the CSA, i.e., with the secure applications within a TEE, the CIA has the potential to send encrypted keys to unlock ECUs to deactivate firewalls, perform decryption of software and allow software updates. *Assumption:* A threat actor controls the CIA and might manipulate installation processes. *Probability: Same as C1. Consequence and Mitigation:* The decryption of keys only takes place inside the TEE; therefore, not visible to the CIA. However, the actual function calls to trusted applications might be manipulated, for instance, by switching encrypted keys between functions. Thus, function calls to the CSA shall be authenticated, e.g., with an authentication tag. We assume an isolated compromise; therefore, the CDA is still intact, and we only have authentic data at local storage [R13].

Threat C3: Consumer Security Agent (CSA). *Location:* Same as C1 and C2. Note that the VUUP file needs to be created for a particular VIN when located in an external tool. *Functionality:* CSA has a TEE and offers a secure execution of cryptographic mechanisms and data. *Assumption:* We assume a complete compromise of CSA. *Probability*: Low, since secured with, e.g., R3, R8 and R13. *Consequence and Mitigation:* Secret keys are compromised. Examples of usage include turning off firewalls, unlocking ECUs for software updates, and software decryption. A secure boot shall protect CSA, and thus, manipulations shall be detected [R3, R8, R13]. However, suppose a threat actor has gained physical control, and the decryption/execution of keys within the TEE is compromised. Keys can be used when connecting to a communication bus to gain extended privileges to ECUs.

However, many ECUs have inherent protection mechanisms such as secure boot and signed software [R8, R12]. Thus, malicious updates will not be approved even when valid diagnostic keys are used to put them in a state for software updates. However, a few legacy ECUs that do not fulfill security requirements might be vulnerable.

7.4.3 Examples of Multilevel Compromise

UniSUF is made to cause the least possible harm when entities are compromised. Each entity shall be implemented as a separate module with the potential to be localized differently (e.g., locally on-prem or containerized in the cloud), along with necessary redundancy. This section will take a few examples of when multiple entities are compromised.

7.4.3.1 The Producer

Threat P1, P2: PDA and PIA. Assume that a threat actor has gained control over the download of data and the diagnostic instructions to install data. ECUs without end-node/secondary validation of signatures might be compromised in that case. However, the PSA is not under attacker control in this scenario. Thus, there will be deviations from the output data from PSA concerning the PDA and the PIA. The PDA, the PIA, and the PSA generate data based on the signed SL file. However, if an attacker can manipulate the normal process flow, e.g., perform ECUs unlock by using authentic data from PSA and then use malicious data for ECUs without end-node protection, these might be compromised [R8, R12]. Post-installation processes shall detect these deviations, and additional updates shall be issued from a redundant non-compromised system [R4,R9,R10].

Threat P1, P2, P4: PDA, PIA and PSA. In the previous multi-level compromise, PSA is not compromised. Thus, cryptographic keys can be sent and processed encrypted to TEE, e.g., to enable extended diagnostic privileges within manipulated installation instructions. Still, the keys are not exposed outside the TEE since they are encrypted. However, if PSA is compromised, we can also assume that secret keys are leaked because PSA can access CMS. In such a case, legacy ECUs without end-node protection are at risk by physical access, e.g., over the OBD-II port. PDA, PIA, and PSA shall all create their data based on the signed SL file. However, if all three entities are compromised, we can assume that the SL file is not enforced. Thus, additional data validation and comparison towards the SL file shall be made on the consumer side, and warning messages issued for deviations [R9,R10]. The primary mitigation is end-node protection, such as using secure boot and signed software [R8, R13]. Legacy systems not adhering to basic security requirements are always at risk.

7.4.3.2 The Consumer

We assume that CDA, CIA, and CSA are localized in the vehicle for this specific case. These can be physically or remotely compromised by, e.g., the driver with the intent to chiptune or by other threat actors to cause harm. Directly compromising entities not part of UniSUF is out of the scope. However, indirect compromises via UniSUF are considered.

Threat C1 and C2: CDA and CIA. A threat actor that has gained control of the CDA can control the process of downloading data. Still, the CSA is not exposed in this case. Thus, the threat actor can only interact with CSA using authentic data since validation of function calls and sent data detects manipulations [R13]. However, a threat actor could manipulate the instructions to download malicious data from other sources. In typical cases, the only way to initiate the installation process is to send authentic installation instructions to the diagnostic client, i.e., CIA [R6]. In this case, the CIA is also compromised but must still interact with CSA using authentic data to start the installation [R13]. As long as CSA is secured, the required keys to perform software installation via UniSUF are not exposed, and no installation can be completed using invalid data. Still, blocking the actual update might be possible [R4,R9].

Threat C1, C2, and C3: CDA, CIA and CSA. In this case, the threat actor has also managed to control CSA. If secret data is leaked from the TEE, such as cryptographic material, we can assume that these keys can be misused for one specific vehicle [R6]. For instance, decrypt software files, disable firewalls and unlock ECUs to enable software updates. Mitigations for these cases are based on end-node protection mechanisms, such as an additional layer for signed software and secure boot [R8, R12]. Critical ECUs, e.g., safety-related, shall always use end-node security.

7.4.4 Comparison to other approaches

Previous solutions such as [278–282] lack a unified approach for the various software update scenarios required within the automotive. Moreover, they do not consider vehicles needing unique updates regarding specific configurations, installed software versions, and unique cryptographic keys. These solutions are missing necessary details, e.g., installation instructions such as handling pre-, peri- and post-state diagnostics, secure transport, and secure execution of ECU-specific cryptographic keys. They also consider changes to all in-vehicle ECUs, which usually is not feasible. UniSUF aims to fill these gaps by proposing a secure and versatile software update framework with previously mentioned considerations in mind.

7.5 Conclusion

Modern vehicles are complex systems containing more than 100M lines of software code controlling various functionality including safety-critical functions and get increasingly vulnerable when adding connectivity. Thus, ensuring timely and secure software updates to patch vulnerabilities is imperative. We have introduced UniSUF, identified entities involved in the distribution and execution during vehicle software updates, provided an attacker model, performed a threat assessment, and elaborated on mitigation mechanisms. We have identified general security requirements for vehicle software updates and mapped them to common security goals and directives further visualized with the Goal Structuring Notation (GSN). The results show that UniSUF fulfills the stated security goals and provides a secure and unified vehicle software update framework that can serve as a detailed reference architecture. We believe our results are valuable not only for automotive software update architects. We also see high relevance for engineers in related areas, such as cyber-physical systems, internet-of-things, and smart cities, guiding the design of secure software update solutions.

Chapter 8

The Automotive BlackBox: Towards a Standardization of Automotive Digital Forensics

Format-adapted version that appeared in IEEE International Workshop on Information Forensics and Security (WIFS) 2023

K. Strandberg, U. Arnljung, T. Olovsson

Abstract. There is a trend toward increased cyberattacks on vehicles. Aligned, forensics requirements and standards are emerging. Digital forensics refers to identifying, preserving, verifying, analyzing, documenting, and finally presenting digital evidence with high confidence in its admissibility, thus ensuring forensics soundness. However, current automotive regulations and standards, such as the United Nations Regulation No. 155 and the International Organization for Standardization standard 21434, provide no details or guidelines. Vehicular data is often extracted using tools unsuitable for digital forensics, thus lacking forensics soundness. The data storage is generally not resistant to tampering and often lacks adequate cybersecurity mechanisms.

Digital forensics is a relatively new field within the automotive domain, where most of the existing self-monitoring and diagnostic systems only monitor safety-related events. To support a forensic investigation, automotive systems must be extended to securely log and store additional information, especially those related to security events. There is no standardization for automotive digital forensics that defines requirements, needed components, and techniques for the automotive domain. In this paper, we identify and propose requirements for automotive digital forensics and present the Automotive BlackBox, an architecture guiding the design of an automotive digital forensic-enabled vehicle.

8.1 Introduction

The complexity of vehicles is increasing at a high pace. A vehicle today can contain around 150 Electronic Control Units (ECUs) and has various connection interfaces, which inherently implies a large amount of data exchange between many entities such as sensors, actuators, ECUs, the Internet, and infrastructure. If this data is assessed satisfactorily through automated processes, a wealth of significant information can be provided to stakeholders such as law enforcement, insurance companies, and manufacturers.

Increased complexity increases the risk of system vulnerabilities and, consequently, the number of potential attack vectors. At the same time, the increased connectivity gives a higher potential to find exploits related to vulnerabilities due to a larger attack surface. For instance, a buffer overflow vulnerability in software (i.e., an attack vector) can be exploited due to an increased attack surface (e.g., a connection interface), enabling the capability to execute arbitrary code and thus potentially disrupt vital in-vehicle functions. Moreover, attacks can be associated with life-threatening hazards due to their potential to affect safety-critical systems such as brakes, steering, and engine control. Thus, such attacks are highly relevant to identifying and tracing in a post-incident digital forensic investigation. It has been shown several times that vehicle cyberattacks have to be taken seriously, e.g., practical attacks in [22, 24] that demonstrates the fragility of automotive systems and the susceptibility to malicious actions to disrupt and modify these systems. For instance, in [24], the firmware was extracted and reverse-engineered to understand hardware features which enabled them to add new functionalities related to their attacks, such as remote access persistence via the cellular connection. Moreover, they managed to add malicious code to a vehicle telematics unit which automatically erased any evidence of its existence after a crash; thus, there was no post-incident available data related to a potentially life-threatening code. Due to the continuous increase in complexity and connectivity, it is only logical to assume that cyber-attacks against vehicles will continue to rise and be even more prevalent. Thus establishing guidelines for forensic automotive design to enable the detection and post-analysis of cyberattacks is imperative.

However, Automotive Digital Forensics (ADF) is a relatively new field within the automotive domain. Most existing self-monitoring and diagnostic systems only monitor safety-related events, such as the status of brakes, seat belts, and airbag deployments, via an Event Data Recorder (EDR). Current vehicle EDRs are used mainly to record limited events under a few seconds before and during a crash, while, e.g., flight data recorders can record hundreds of parameters for many hours. Numerous vehicle manufacturers already transmit EDR-related data to a central location, such as the GM´s OnStar, in the occurrence of a crash [24]. To support a forensic investigation, these systems must be extended to log additional information, especially cybersecurity-related, e.g., cyberattacks. A satisfactory ADF solution must consider in-vehicle data and its surroundings from an individual, vehicle fleet, and infrastructure perspective.

In [4], four main stakeholders are identified for ADF, namely: Law Enforcement (LE), Vehicle Manufacture (VM), Vehicle Drivers (VD), and Insurance Companies (IC). LE refers to, e.g., the police and related legal systems. VM requires ADF data for fault-tracing, e.g., to distinguish hardware and software failures with non-malicious origin from cybersecurity incidents, e.g., attacks from threat actors. VD might try to remove or manipulate forensic evidence with the intent to hide traces of crime, whereas IC are interested in insurance cases and cost and risk profiling/statistics. Stakeholders, e.g., LE, IC, and VM, must establish a trustworthy and admissible chain of events to derive the cause of accidents concerning malicious actors, e.g., hackers and terrorists, and non-malicious actors/origins, e.g., weather and animals on the road.

Contributions. We present the Automotive BlackBox, an architecture guiding the design of an automotive digital forensic-enabled vehicle. We highlight challenges, identify forensic components, and propose a standard data format, techniques, goals, and requirements in an architectural automotive context, considering current and upcoming regulations and standards. Based on our previous work, a systematic literature review of the area [4], our contributions are novel and relevant for automotive digital forensics investigations.

8.2 Challenges

Quite a few challenges need to be considered when establishing requirements for ADF. A modern vehicle consists of many devices running various operative systems. Furthermore, they communicate over many different protocols internally and with the outside world via Vehicle-2-everything (V2X) communication. An immense amount of data is continuously transmitted, e.g., with safety-critical systems, including brakes, steering, and acceleration. In previous work, we identified 16 categories of forensically relevant data and stated the required security properties for the data [4]. However, modern vehicles only log a fraction of forensically relevant data, and manual approaches are often used to manage the data.

A vehicle has various devices where the data is spread out in multiple places in a distributed fashion, e.g., different ECUs, networks, and the cloud. Locating all devices containing relevant data is challenging since the vehicle's proprietary architecture. Data can be stored in Virtual Machines (VMs), where data in registry entries and temporary files can be erased when turning off or rebooting the machine. Thus, there is a need to extend and automate data collection covering all relevant data. However, currently, there are no standardized interfaces for information extraction and no standardized format for storage. For instance, sometimes desoldering memory chips are required to extract data. Moreover, forensic investigations require following an established process, a scientifically proven methodology, and using validated tools and techniques to maintain the chain of custody, but that is currently only sometimes the case since the lack of standardization within ADF forces OEMs to develop and use their tools and strategies for fault tracing and data collection.

Manual approaches for managing the steady increase in forensically relevant data, considering data collection, extraction, and analysis, are time-consuming. Sufficient pertinent data needs to be improved, and the security mechanism needs to be more robust in ensuring trustable data. For instance, many legacy systems and protocols currently lack satisfactory security features. In many cases, there is a need for more performance, better storage capacity, and increased data security to enable a reasonable level for ADF. The related cost of fixing these issues is challenging. There is also multi-jurisdictional litigation to consider, sometimes contradictory, e.g., privacy regulations [9] versus requirements for data collection for forensics investigations [8]. There are requirements to secure data [8], which makes data availability for forensics challenging due to the inaccessibility of secret keys for decryption. Moreover, security techniques required for forensics might negatively affect requirements for safety-related time-critical systems [5]. Thus, requirements for privacy, forensics, cybersecurity, and safety regulations sometimes conflict.

In summary, we conclude the following main challenges: (i) Only a fraction of logging and analysis is currently performed on available data. Moreover, data is spread out in various places making identification and retrieval time-consuming. Thus, an increase in automated data collection for all relevant data is needed. (ii) Due to, e.g., cost and performance restraints in current vehicles, trustable data is often not ensured, nor are common security properties fulfilled for digital evidence. Thus, there are requirements to secure potential evidence better. (iii) Regulations, standards, and common guidelines within ADF concerning, e.g., forensics processes, data collection, management, formats, and tools must be evolved and revised. Thus, there is a necessity to standardize ADF to ensure forensic soundness. (iv) Regulations in different fields and countries must be revised to align, i.e., privacy, forensics, security, and safety. Variations of forensic solutions must be considered in different countries. The following sections consider challenges i-iii and leave iv as further work.

8.3 Digital Forensics Principles

Digital forensics is a field with strong dependencies on information security. It is imperative to ensure available trustable data. Although digital forensics mainly emphasizes post-incident, cybersecurity aims to mitigate threats to forensic data, such as removing and manipulating digital evidence.

Digital forensics includes the collection and investigation of data, generally about crime. Security techniques must be used for the data to be admissible in a court of law. Generally accepted principles of a digital forensic investigation apply to ADF. However, the naming and number of steps might differ between methodologies, although the core concepts are usually the same [4], namely: (i) *Identification.* Has a crime occurred? What data is relevant, and where is the data stored? What resources (e.g., tools and experts) are needed? (ii) *Preservation.* How can we preserve data integrity (e.g., running devices, remote access, extraction, and anti-forensics)? (iii) *Acquisition and verification.* How can we extract (e.g., imaging, log files, live acquisition) and validate the data's authenticity (e.g., signatures and hashes)? (iv) *Analysis.* What type of information is relevant to assess? (v) *Reporting.* How can we document all parts of the forensic investigation and its related result to be admissible in a court of law? Moreover, law enforcement guidelines need to be considered, such as the four Association of Chief Police Officers (ACPO) principles [283].

A forensic investigation requires establishing trust in the chain of events, where the life cycle of the data must be considered. Thus, processes for handling forensic data are needed as technical solutions to collect and secure forensics data. Any inadequacies in these two can potentially devastate the forensic case, e.g., making the data invalid for the investigation.

8.4 The Automotive BlackBox

As mentioned in Sections 8.1 and 8.2, the forensic mechanisms of today's vehicles, e.g., EDRs, are insufficient for ADF. A more comprehensive approach is necessary to align with current and upcoming standards and regulations. In the remainder of this section, we state an attacker model, Automotive Digital Forensics Goals (ADFG), requirements, technical details, and a reference architecture for ADF.

Attacker Model. We consider the six threat actors as stated in [276], namely, the Financial Actor (FA), the Foreign Country (FC), the Cyber Terrorist (CT), the Insider (IN), the Hacktivist (HA), and the Script Kiddie (SK). We assume a common agenda to perform various cybercrime targeting vehicles with the potential to affect the driver, passengers, and objects in the vicinity using the vehicle as leverage. However, the main objective is to hide, delete or manipulate digital evidence, such as digital traces of crimes, to obstruct or prevent forensic investigation.

Automotive Digital Forensics Goals and Requirements. Based on the attacker model and previously mentioned principles and challenges, we establish six ADFG. ADFG-1 is a general rule based on the availability and trust of digital evidence, and ADFG-2 to ADFG-6 are more specific based on accepted forensic principles. Challenges are summarized at the end of Section 8.2, where ADFG-1 assesses challenges (i) and (ii), and ADFG-2 to ADFG-6 assess challenge (iii). With the attacker model in mind, these ADFG are further mapped into specific requirements inspired by a standardized approach for argument notation to demonstrate coverage [277] (cf. Table 8.1 and Figure 8.1). For instance, ADFG-1 is general and about *Availability and Trust* and needs R1-R7. ADFG-2 is more specific and concerns *Data Identification* and needs R2, R4-R6, and R9-R10. As emphasized in previous work by us [4], we adopt the well-known *CIA* security triad extended with two other properties, *NP*, where the first four are prerequisites for securing vehicle forensic data and the fifth for personal data. An explanation of these properties follows. *Confidentiality(C)* guarantees that only authorized entities can access and disclose data. *Privacy(P)* concerns personal data, such as traffic violations, location data, and synced data from external devices, e.g., text messages and phone records. Therefore, such data must be protected according to local laws and regulations [9]. *Authenticity* is a form of *integrity(I)* ensuring data origin and is imperative for forensic investigations. *Availability(A)*, e.g., in the event of a crash, must be ensured, and secure and tamper-proof storage guaranteed. *Non-repudiation(N)* ensures that occurrences of events and their origin can not be denied. Therefore, *authenticity* and *integrity* are required for *non-repudiation.* An explanation of the six ADFG follows.

ADFG-1: Availability and Trust of Digital Evidence. A prerequisite for ADFG-2 to ADFG-5 is available and authentic data. Thus, we identify requirements for technical solutions to detect and securely store forensically relevant events, including fulfilling R1 (cf. Table 8.1) and the CIANP properties for digital evidence where applicable. *ADFG-2: Identification.* The first step is to

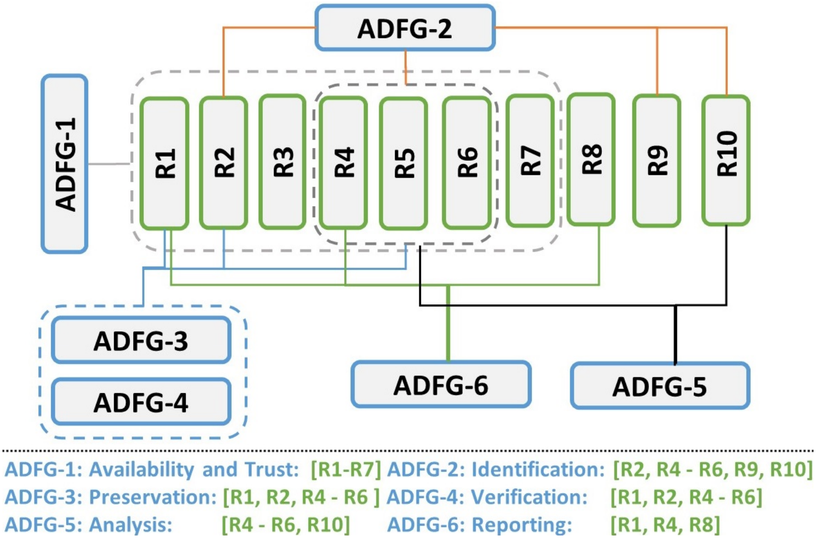

ADFG-2
ADFG-1
R1
R2
R3
R4
R5
R6
R7
R8
R9
R10
ADFG-3
ADFG-4
ADFG-6
ADFG-5
ADFG-1: Availability and Trust: [R1-R7] ADFG-2: Identification: [R2, R4 - R6, R9, R10]
ADFG-3: Preservation: [R1, R2, R4 - R6] ADFG-4: Verification: [R1, R2, R4 - R6]
ADFG-5: Analysis: [R4 - R6, R10] ADFG-6: Reporting: [R1, R4, R8]

Figure 8.1: Mapping of Automotive Digital Forensics Goals to Requirements

identify what has happened. Has there been a crash? Can anyone describe the incident? What is the most relevant data to assess? To identify evidence, a prerequisite is satisfactory data collection and filtering. An Intrusion Detection System (IDS) shall, therefore, detect and securely store events related to anomalies and predefined patterns. Moreover, other forensically relevant events shall be considered. For instance, time for braking, acceleration, seat-belt traction, airbag deployment, weather conditions, location data, and detected warnings (e.g., tired driver, lane assist, V2X data) can all be relevant to contribute to establishing the cause of an incident. *ADFG-3: Preservation.* How can we guarantee integrity and privacy during data collection? How can we ensure that relevant data is recovered? Can the engine be turned off? Is there a risk that data can be erased by a perpetrator remotely? Potential evidence shall be stored securely, considering the CIANP properties. *ADFG-4: Verification.* How can we validate the authenticity of the data? Evidence shall be stored in a standard format that includes the potential to validate time, integrity, and origin. *ADFG-5: Analysis.* What data is relevant to assess concerning the crime investigated? Forensic data shall be identifiable concerning the type of data and the order of occurrence. For instance, detecting anomalies in the network aligned with normal events such as opening and closing doors, speed, braking, and location data. Data collection and analysis shall be automated, and manual work reduced to a minimum. For instance, incorporate Artificial Intelligence (AI) and Machine Learning (ML) approaches for automated data management. *ADFG-6: Reporting.* How can we document the evidence and ensure admissibility in legal proceedings? It shall be possible to identify relevant data for a predefined period concerning the type and order of events in relation to a potential crime. Data shall be verifiable concerning authenticity with a detailed timeline for the events.

Table 8.1: ADF Requirements

Requirement R1: fulfilment of CIANP. R1a. Confidentiality. R1b. Integrity. R1c. Availability. R1d. Non-Repudiation. R1e. Privacy

Requirement R2: secure logging, storage and extraction. R2a. There shall be mechanisms that guarantee the authenticity of logged and stored data. R1 and mechanisms for preventing data modification, tampering, and deletion shall be considered. R2b. Storage shall be constructed with physical integrity in mind, thus, to survive crashes and physical violence. R2c. Forensically relevant events shall be securely stored for fault tracing and post-incident investigations. For a list of relevant data to consider, we refer to [4]. R2d. A secure physical extraction interface shall exist, requiring mutual authentication to extract forensic images.

Requirement R3: infrastructure and communication. The infrastructure, cryptographic algorithms, and key material shall follow best security practices.

Requirement R4: common format and tools. Forensic data shall have a common format. The format shall be verifiable and contain information about the logical order of occurrence. The tools used shall adhere to standardized, accepted, and regulated digital forensics processes.

Requirement R5: time. It shall be possible to trace the logical order for events according to a time value, e.g., the logical and clock time. Thus, the forensic system requires trust in a time server and an agreement on the logical order of events.

Requirement R6: redundancy. Relevant redundancy shall be used to ensure that data is authentic and available. For instance, the same data stored in different sources, such as in-vehicle and cloud data, shall be possible to verify its identical and detect deviations.

Requirement R7: secure boot. State-of-the-art secure boot protection mechanisms shall be used where applicable, e.g., manipulations in relevant entities, such as the IDS, SIEM, and the Automotive Blackbox, shall be detected.

Requirement R8: least privilege. Data shall only be available to authorized entities.

Requirement R9: Intrusion Detection/Prevention Systems. IDSs/IPSs shall detect and react to anomalies from normal communication patterns and known attacks, e.g., maintain secure logging of relevant events (R2).

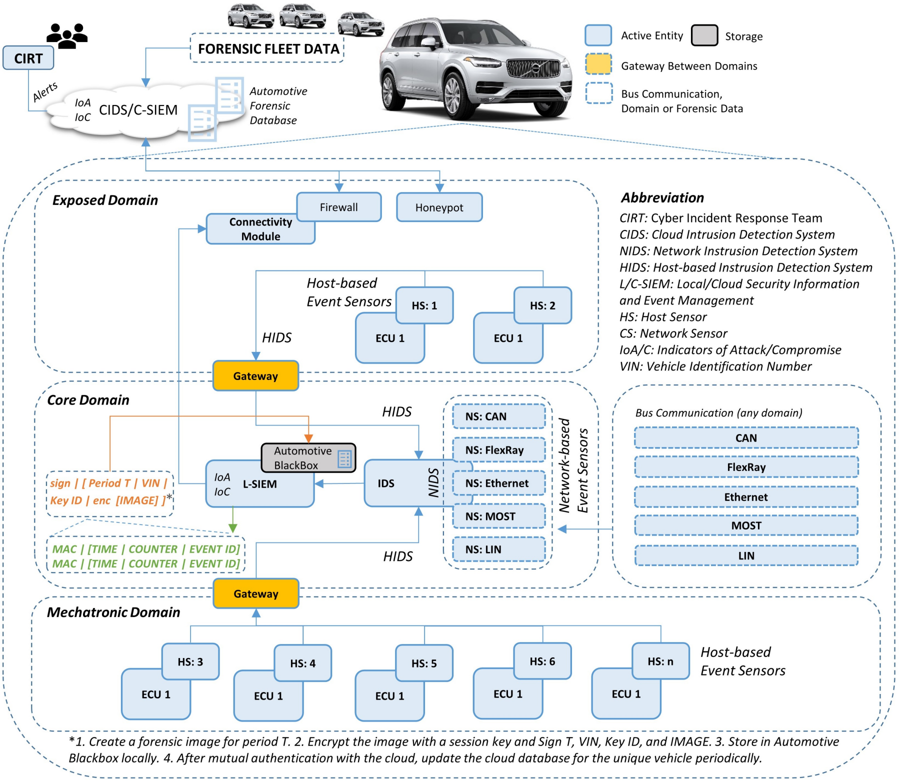
Requirement R10: threat intelligence. Learning about attacks to keep pace with attackers shall be possible, for instance, using honeypots and analyzing, correlating, and mapping data from multiple sources.

8.4.1 Technical details

IDS shall detect *Indicators of Attacks (IoA)* and store *Indicators of Compromise (IoC)*. The main distinction between IoAs and IoCs is that the former are ongoing events of potential attacks, while the latter are events indicating a previous compromise. Thus, one or many IoAs can give rise to IoCs, where the latter is most relevant from an ADF perspective. IDSs can record anomalies from a predefined pattern (anomaly-based IDS) and detect specific signatures (specification-based IDS). The former is more suitable for detecting unknown attacks, and the latter is better at detecting known attacks. A higher rate of false positives is usually the case for the former and false negatives for the latter. Thus, a hybrid approach is beneficial to increase coverage. IoCs from IDS can be forensic evidence of potential network and ECU breaches, e.g., unusual traffic and other deviations. An example can be that the speed should always be zero when the vehicle is in parking mode. Any mismatch in specific signals or vehicle status can indicate IoAs or IoCs. Other examples are failed authorization attempts (e.g., attempted access of privilege mode via debug ports), invalid software signatures during updates, or the secure boot process.

IoCs and IoAs from IDSs are managed by a *Security Information and Event Management (SIEM)* along with other detected relevant events, such as safety-related, e.g., braking, acceleration, steering, engine control, airbag release, and seat belt traction. Examples of non-safety-related events are software update events, location, opening/closing of doors, and executed diagnostics. V2X communication with infrastructure, other vehicles, and external devices can be forensically relevant. We do not aim to provide a complete list of forensically relevant data. Still, we refer to our previous work [4], which identified relevant ADF data. SIEM offers real-time monitoring, analysis, data collection, and storage of events and logs from various sources. It creates an in-depth overview of previous and ongoing events for threat management and auditing purposes, e.g., threat mitigation, fault tracing, and ADF. AI and ML approaches automate management, e.g., rating alerts. SIEM data and automated analysis are further transferred to a Cyber Incident Response Team (CIRT) for further analysis and decision-making.

8.4.2 Architecture



Abbreviation
CIRT: Cyber Incident Response Team
CIDS: Cloud Intrusion Detection System
NIDS: Network Instrusion Detection System
HIDS: Host-based Instrusion Detection System
L/C-SIEM: Local/Cloud Security Information and Event Management
HS: Host Sensor
CS: Network Sensor
IoA/C: Indicators of Attack/Compromise
VIN: Vehicle Identification Number

**1. Create a forensic image for period T. 2. Encrypt the image with a session key and Sign T, VIN, Key ID, and IMAGE. 3. Store in Automotive Blackbox locally. 4. After mutual authentication with the cloud, update the cloud database for the unique vehicle periodically.*

Figure 8.2: The Automotive BlackBox within a centralized architecture context

We propose a core architecture with domain separation according to Figure 8.2, where hybrid IDS components detect and securely log events. As shown, sensors can detect specified ECU events and anomalies in communication, further sent to SIEM for automatic analysis and secure storage according to a predefined format. We propose to use a hybrid SIEM, divided into a local (L) and cloud (C) part, adaptable where analysis occurs dependent on performance and cost restraints. For instance, L-SIEM can be implemented with measures during cyberattacks, e.g., log and analyze, while others are offloaded to C-SIEM, which takes further decisions and generate fleet responses. Another option, if performance/cost is an issue, is to run the local part in log-only mode, i.e., only log events and create images for a defined period. C-SIEM and the CIRT entirely perform the analysis for the latter case.

The L-SIEM stores the events in the correct order in the Automotive BlackBox, including time, and a counter that keeps track of the number of occurrences for each event. A pre-shared certificate between C-SIEM and L-SIEM can be used for the key-wrapping of symmetric keys to ensure secure storage/transfer. For such a case, L-SIEM can create a list of symmetric keys further used to encrypt images of forensic data. In turn, keys are encrypted with an asymmetric public key from the public part of the shared certificate. Only C-SIEM and the CIRT team can access the corresponding private key. Thus, the required symmetric keys are kept in escrow to protect user privacy. For instance, if a malicious entity manages to extract or manipulate digital evidence stored in the Automotive BlackBox, it is still encrypted, ensuring confidentiality/privacy, and signed, ensuring integrity/non-repudiation.

In summary, we propose that L-SIEM use the public part of a pre-stored encryption certificate and the private part of a pre-stored signing certificate, where the C-SIEM has access to the corresponding part for decryption and validation. Thus, any authorized entity, such as a forensic investigator, must request the symmetric decryption keys from the CIRT to access and disclose potential digital evidence. The L-SIEM securely, i.e., encrypted and mutually authenticated, uploads forensics data, i.e., the image, to the cloud for a specified time interval and a.s.a.p. if a vehicle is out of range from connectivity. The C-SIEM verifies the image signature before storage. The C-SIEM and the *Automotive BlackBox* have identical redundant data for a defined time. Thus, concerning that period, it can be compared for potential deviations.

Due to cost restraints within the automotive, we propose using a circular memory buffer, i.e., a first-in-first-out (FIFO) approach where old periods are overwritten by new periods. The downside is that data might be lacking beyond that period, for instance, due to a long time without connectivity. Additionally, as shown in Figure 8.2 we propose using a forensic honeypot, which attracts attackers to learn about their intentions and attack types to analyze, investigate and mitigate future attacks. Honeypots must be adapted regularly, e.g., via secure software updates [18, 19], to lower the risk that threat actors learn it's not a real system. We propose that relevant events are analyzed to acquire a status of the vehicle fleet's health, such as awareness of ongoing large-scale cyberattacks. We acknowledge the cost constraints within the automotive industry, and although beneficial, honeypots and similar solutions might not always be feasible. Also, note that our solutions cover mainly forensically relevant events from the ECUs and communication buses in the vehicle. However, specific synced data from external devices (cf. Table III in [4]) can contain relevant but privacy-sensitive data, such as messages and call logs. Our approach does not cover automated retrieval and analysis of such data, but our proposal can be extended by transferring it to the Automotive BlackBox and further to C-SIEM for processing. However, aligning with local laws and regulations concerning privacy-sensitive data is important. Moreover, we acknowledge that some ECUs might not be able to have a host-based sensor and still contain relevant data. Still, our approach aims to automate the data collection and analysis process as much as possible to limit manual work.

Standard Data Format and Key Management. We propose the format as visualized in Figure 8.2, which contains the following attributes for each event. MAC is a key-based cryptographic hash over the rest of the event values. TIME, real or logical time, depends on the available source to synchronize time between different devices in the vehicle. COUNTER the number of occurrences of the same events under a predefined period. EVENT ID, the identification number of the actual event taking place.

L-SIEM creates an image for a predefined period of events, generates a symmetric key, and encrypts the image with this key. Identification data for a period T, VIN, and encryption key ID (not the actual key) is added to the image metadata, whereafter, a hash is calculated over the data and signed with an in-vehicle pre-stored certificate. The symmetric key used to encrypt the image is further encrypted with the public part of a pre-stored certificate in the vehicle and added to a key manifest along with the key ID. Key manifest is stored with encrypted images in the Automotive Blackbox, further periodically synchronized with and stored in the cloud. The private part of the certificate is securely stored and accessed at C-SIEM to decrypt symmetric keys for further decryption of image files enabling automatic direct analysis by CIRT.

8.5 Discussion and Future Work

Infrastructure development differs in countries and locations, where cost and transfer speed can be challenging. There might be storage limitations in the vehicle, where a satisfactory storage size might be too costly. Low storage means that only a limited time can be saved in-vehicle. Constraints in connectivity, transfer speed, and cost might lead to that important data can be lost. As previously mentioned, there can be many distributed ECUs and sensors, and ensuring the authentic order/timing of events is challenging. Entities might suddenly stop generating alerts and must be detected. For instance, units can be disabled by hardware failures originating from malfunction or cyberattacks. However, having, e.g., a heartbeat signal from devices might not be possible due to performance restraints. From a fleet perspective, it is valid to be able to correlate time between events, for instance, speed, acceleration, and braking between involved vehicles, something that C-SIEM can automate. Enabling the collection of potential digital evidence and still adhering to privacy regulations is difficult. Data might reveal sensitive information about other individuals than intended via external communication or when correlating data. Anonymizing data and, at the same time, being able to connect it to individuals potentially involved in a crime is both contradictory and challenging.

Using AI, ML, and blockchain technology in automated data collection and analysis is promising for future research. Challenge iv (cf. Section 8.2) aligning and revising different regulations and standards, i.e., privacy, forensics, security, and safety, emphasizing ADF is important, as studying the cost impact of new architectures. The chain of custody needs to be fulfilled to ensure forensic soundness. Trust in keys for encryption, signing, and MAC values is imperative to guarantee the CIANP properties. More work is needed to analyze potential attack vectors, e.g., vulnerabilities in key management, process isolation, and virtualization technologies such as trusted execution environments and containers.

8.6 Related Work

In 2006, NIST released SP 800-86, a document for practical guidance on performing computer and network forensics. SP 800-86 defines digital forensics as the science of identification, collection, examination, and analysis of data while preserving the integrity of the information and maintaining a strict chain of custody. The ISO 27037, yet another standard for digital forensics, was established in 2012 and further reviewed and confirmed in 2018. ISO 27037 provides digital evidence identification, collection, acquisition, and preservation guidelines. In 2004, NIST published SP 800-72 (PDA Forensics), and later in 2007, SP 800-101r1 (Mobile Device Forensics) provided guidelines for tool usage and procedures about PDAs and mobile devices. However, these documents are not automotive-specific, thus, do not provide satisfactory guidance within this area. In [4], we introduce the area of ADF. We perform an extensive systematic literature review where we consider over 300 publications. We further group relevant papers into surveys, technical solutions, and focus categories. We also assess the cybersecurity aspect of the technical solutions by discussing and mapping them to cybersecurity attributes where applicable. Furthermore, we detail the type of forensically relevant data mentioned that was considered and how it needs to be secured. However, to the best of our knowledge, there is no previous work that extensively details goals and general requirements in an architectural context with the aim to guide ADF design with current and upcoming regulations in mind. Thus, our contributions are both novel and important.

8.7 Conclusion

We have introduced the Automotive BlackBox, an architecture for automotive digital forensics, including components, standard data format, techniques, goals, and requirements. We have identified and highlighted challenges, such as the lack of existing regulations, standards, and common guidelines, and considered them when establishing our architecture. The identified goals are inspired by accepted digital forensics principles and have been further mapped to specific automotive requirements via a standardized approach for argument notation to ensure broad coverage. Furthermore, we have presented detailed guidelines, including a conceptual architectural description, key management, and data formats. Considering current and upcoming regulations, our contributions are useful in guiding the design of automotive and similar systems within a digital forensics context.

Chapter 9

Towards a Formal Verification of Secure Vehicle Software Updates

Format-adapted version in submission

M. S. Hagen, E. Lundqvist, A. Phu, Y. Wang, K. Strandberg, E. M. Schiller

Abstract. With the rise of software-defined vehicles (SDVs), where software governs most vehicle functions alongside enhanced connectivity, the need for secure software updates has become increasingly critical. Software vulnerabilities can severely impact safety, the economy, and society. In response to this challenge, Strandberg et al. [31] introduced the Unified Software Update Framework (UniSUF), designed to provide a secure update framework that integrates seamlessly with existing vehicular infrastructures.

Although UniSUF has previously been evaluated regarding cybersecurity, these assessments have not employed formal verification methods. To bridge this gap, we perform a formal security analysis of UniSUF. We model UniSUF's architecture and assumptions to reflect real-world automotive systems and develop a ProVerif-based framework that formally verifies UniSUF's compliance with essential security requirements — confidentiality, integrity, authenticity, freshness, order, and liveness —demonstrating their satisfiability through symbolic execution. Our results demonstrate that UniSUF adheres to the specified security guarantees, ensuring the correctness and reliability of its security framework.

9.1 Introduction

Connected cars are quickly becoming the norm, with 96% of manufactured cars in 2030 expected to have connectivity features [284]. These connected cars feature a multitude of electronic control units (ECUs), with more than 100 ECUs per vehicle [285]. Maintaining and regularly updating these ECUs is critical to prevent security vulnerabilities. The massive scale of the automotive industry, with over 70 million cars sold worldwide annually [286], makes it a prime target for malicious actors. If an attacker compromises the software update process, the result can be malware installation, sensitive data leakage, and even vehicle hijacking, leading to devastating financial, social, and potentially fatal consequences.

Despite the importance of secure updates, ensuring the confidentiality, integrity, and correct execution of the software update process for connected vehicles remains a highly challenging task. In particular, attackers could exploit weaknesses in the update process to install malicious software, eavesdrop on sensitive information, or revert vehicle software to an obsolete version containing vulnerabilities. Moreover, the sheer number of vehicles and ECUs in each car presents a scalability challenge, complicating the implementation of robust security measures across the entire fleet.

9.1.1 Existing Solutions and Their Shortcomings

Frameworks have been proposed to address these challenges, including the Unified Software Update Framework (UniSUF) [31]. UniSUF proposes a reference architecture intended to serve as input to standards for secure software updates. UniSUF's specifications were driven by the following development goals [19].

Confidentiality: To hinder eavesdropping.

G1: Ensure software confidentiality during the software update process.

G2: UniSUF session keys can only be viewed in decrypted format by authorized software components.

Integrity and Authenticity: To hinder spoofing and tampering.

G3: The software is authentic against a certificate and remains unchanged during the update process.

G4: Only authentic resources are processed.

Freshness: To hinder replay attacks.

G5: An adversary should be unable to revert a vehicle's software to a previously installed version.

G6: The system creates unique software distribution files per software update. Each such file can only be used for a designated vehicle.

Order: To hinder vulnerabilities that take advantage of running the process in an unintended order.

G7: The software update process should follow the correct order.

Liveness: Hinder DoS attacks.

G8: The software update process should eventually terminate, regardless of success or error.

UniSUF's prior evaluations focus primarily on practical deployments and lack formal verification of its security guarantees. This leaves the potential for subtle vulnerabilities that attackers could exploit, notably by exposing cryptographic keys or violating the sequence of operations during the update process.

Several scientific challenges are associated with the formal verification of systems like UniSUF. These include [**1: Confidentiality**] ensuring that the software update process maintains the confidentiality of secret information throughout execution; [**2: Integrity and Authenticity**] verifying that no unauthorized modifications occur during the update process; and [**3: Order and Liveness**] guaranteeing that the update process follows the correct sequence of actions and terminates appropriately. These challenges are compounded by the complexity of modeling such systems in formal verification tools such as ProVerif [287, 288], which requires a precise representation of the security assumptions and the adversary model.

The challenges associated with the formal security of UniSUF raise the following research questions.

RQ1: UniSUF has certain secrets, essential for its operation, for example, cryptographic keys and disseminated software. The question is whether UniSUF's operation might expose any of its secrets.

RQ2: How can we guarantee that the software that UniSUF disseminates is obtained from an authentic source and not manipulated?

RQ3: How can we guarantee that in UniSUF, it is impossible to perform a software update with obsolete software versions?

RQ4: Appropriate software updates require a specified order of actions. How can we guarantee that UniSUF's update operation proceeds orderly?

RQ5: How do we know that the software update process always ends?

9.1.2 Our Contribution

We address the above challenges and research questions by conducting a formal security analysis using symbolic execution in ProVerif. Our key contributions to advancing the state of the art are as follows:

- We model UniSUF's architecture (Section 9.5) and assumptions (Section 9.3) to reflect real-world automotive systems. This model is represented in ProVerif (Section 9.6) to show the formal satisfiability of essential security properties, including confidentiality, integrity, authenticity, freshness, order, and liveness.

- We introduce novel techniques (Section 9.7) to simulate UniSUF in a symbolic execution environment within ProVerif, allowing us to formally analyze critical aspects, such as software authenticity and the correct sequence of operations.

- Through formal proofs and experimental results, we ensure that UniSUF's update process terminates and follows the correct procedural order.

The primary outcomes of our results are as follows.

- A rigorous formulation of UniSUF's security requirements, emphasizing confidentiality, integrity, authenticity, freshness, order, and liveness in the software update process.

- An open-source ProVerif-based framework to formally verify UniSUF's compliance with these security guarantees.

- Through ProVerif's symbolic execution environment, we demonstrate that UniSUF can satisfy the proposed security requirements under realistic adversary models while considering a system architecture that represents real-world deployment.

Our results show that UniSUF's architecture effectively prevents attacks, such as secret exposure and replay attacks while ensuring that software updates proceed in the correct sequence and terminate as expected. This demonstrates that UniSUF's architecture, assumptions, and security requirements can be formally satisfied, providing strong assurances for its security in real-world deployments (Section 9.8).

To ensure the reproducibility of our results and to encourage further development, we pledge to release our solution as open-source upon acceptance of the paper, as detailed in Appendix (Section 9.9).

9.2 Related Works

Analyzing the security of a complex system must take into account potential threats. Strandberg et al. [30] proposes a security and resilience framework, i.e., Resilient Shield, utilizing a security enhancement methodology [29] along with mitigation mechanisms from [13] in light of an analysis of attacks targeting vehicles. Based on the identified attacks, the authors establish security goals and specify the directives required to achieve them. In [19], Strandberg et al. detail specifications for meeting security goals with an emphasis on vehicle software updates. The authors provide a detailed threat analysis of UniSUF for different threat scenarios, aligned with security goals [29], and detailed requirements for the UniSUF architecture [31].

There are formal verification tools that can either assist or automate proofs. For instance, we can express systems and their properties as logical formulas and use theorem provers such as Coq [289] and Isabelle [290] to either interactively or automatically prove the specified properties. However, formulating the entire implementation of complex systems is tedious and error-prone. Instead, it can be more intuitive to verify a system based on a formal model using model checkers such as Spin [291], Uppaal [292], and TLA+ [293]. Model checkers take a formula and a model and then verify whether the formula holds within the model. The appropriate model checker can simplify the process of deriving a system model. For example, Uppaal is a model-checking environment that provides both a graphical interface and a modeling language to model real-time systems [292], which makes it well suited for time-critical systems.

For our purposes, we also need to model the adversary, to be explicitly defined in general model checkers. In contrast, cryptographic protocol verifiers, such as ProVerif [288], CryptoVerif [294], and Tamarin Prover [295], are designed to focus on security and implicitly incorporate the adversary model. For instance, these cryptographic protocol verifiers verify their models under the assumption of the Dolev-Yao adversary model [296]. Therefore, cryptographic protocol verifiers are more appropriate for security properties as they eliminate the need to model an adversary.

In [297], Wang provides protocols to attest that the manufacturer has approved vehicle hardware. These protocols enable the replacement of old components with new ones and the attestation of all vehicle components during start-up. Wang used formal methods to prove the correctness of his protocols. He defined the system model, its assumptions, and requirements and used ProVerif [288] to formally verify that his protocols fulfill the specified requirements given the system model and assumptions. Wang's work has been an inspiration for our own research in this area.

In [298], Basin et al. used Tamarin Prover to find weaknesses in the Authentication and Key Agreement protocol used by 5G. Tamarin Prover has also been used to analyze WiFi Protected Access 2 [299] and Transport Layer Security 1.3 (TLS 1.3) [300]. In [301], an analysis of TLS 1.3 was performed with ProVerif [288], where they also looked at a privacy extension for TLS 1.3 called Encrypted Client Hello.

In addition, there has been research related to formal verification of software update protocols. In [302], Ponsard and Darquennes conducted a survey of various over-the-air update protocols and conducted a case study on formal verification of the UpKit protocol, a software update framework designed for IoT devices with limited resources [303]. The case study demonstrated how they modeled and verified the UpKit protocol using the Tamarin Prover. However, they did not create a theoretical model for UpKit to showcase the reasoning about update protocols, and neither did they specify their Tamarin model's underlying assumptions and requirements.

9.3 Preliminaries

We provide our definitions, assumptions, and requirements.

9.3.1 System Settings

The system consists of computing entities that interact through communication channels. We assume the system is synchronous and that all entities can access universal time. Every entity has a state, including its variables and all messages in its incoming communication channels. The entities update their states by taking atomic steps. Each step performs an internal computation that takes one time unit. These steps can also receive or send messages. An unbounded sequence of atomic steps, X, denotes an execution. For a given entity E, an execution of E is a subsequence of X from which all steps not taken by E are omitted. Each of our studied problems is solved by a distributed algorithm. The system entities collectively execute an algorithm by individually running a sequence of tasks. Each problem is divided into the sub-problems we analyse in Section 9.6. The last task in each sequence is the *halt* task.

9.3.2 Threat Model

Based on the Dolev-Yao model [296], the adversary has complete control over the communication between entities. In addition, message interception, injection, and modification are also possible. The adversary may also delay the delivery of the message by a bounded time η; therefore, communication channels are assumed to be reliable but without guarantees of FIFO ordering. This is derived from Wang [297].

9.3.3 Cryptographic Primitives, Notations, and Assumptions

We assume access to the standard cryptographic primitives in Table 9.1. We emphasize the requirement for an authenticated symmetric encryption scheme, such as AES-GCM, which ensures that the encrypted data remain confidential and are authenticated to verify the sender [304]. This is necessary because some secrets must be authenticated, such as inputs to the Trusted Execution Environment [19, Tab. 1]). To simplify our model, we define a certificate as valid if it is signed by the root certificate. For simplicity, we omit the explicit notation of these signatures in our cryptographic descriptions, assuming that all valid certificates are implicitly signed. UniSUF operates under the assumption of a trusted root certificate [19, 31]. Consequently, we assume that this root certificate is securely pre-installed in the vehicle.

UniSUF assumes secure and reliable communication between entities. This can be achieved, for example, by using SSH [305] or mutual TLS [306, 307]. Therefore, adversaries cannot eavesdrop, tamper, or replay messages, with the exception of one link (see Section 9.6.3.5) for which UniSUF uses reliable non-FIFO communication without security guarantees. UniSUF uses cryptographic materials (see Section 9.4.1), such as symmetric keys and cryptographic signatures.

Notation	Description
Obj_{Key}	Symmetric key of type Obj.

Continued on next page

Table 9.1: Notations used in the communication schemes. Inspired by [297, Table 1].

Notation	Description
E_{Cert}	Entity E's certificate is an asymmetric key pair. The key pair's public key is signed by the root certificate and only E knows the private key. The private key is omitted when E_{Cert} is included in a data structure or message.
$E_{PrivateKey}$	The private key belonging to E_{Cert}.
$E_{PublicKey}$	The public key belonging to E_{Cert}.
$AsymEnc(Message, E_{PublicKey})$	Asymmetrically encrypts the given $Message$ with $E_{PublicKey}$, thereby creating $CipherText$.
$AsymDec(CipherText, E_{PrivateKey})$	Asymmetrically decrypts the given $CipherText$ with the private key $E_{PrivateKey}$, thereby recovering $Message$.
$SymEnc(Message, Obj_{Key})$	Symmetrically encrypts the given $Message$ with the symmetric key Key, thereby creating $CipherText$.
$SymDec(CipherText, Obj_{Key})$	Symmetrically decrypts the given $CipherText$ with the symmetric key Obj_{Key}, thereby recovering $Message$.
$AuthSymEnc(Message, Obj_{Key})$	Encrypts the $Message$, similar to $SymEnc(Message, Obj_{Key})$, but will also include an authentication tag that hinders $Message$ from being changed, and validates if the Obj_{Key} is used to encrypt $Message$.
$AuthSymDec(CipherText, Obj_{Key})$	Decrypts the $CipherText$, similar to $SymDec(CipherText, Obj_{Key})$, but any change to the encrypted message or any message encrypted by $Obj'_{Key} \neq Obj_{Key}$ will be detected.
$Hash(Message)$	Creates a hash H of $Message$, such that $Message$ cannot be retrieved from H.

Continued on next page

Table 9.1: Notations used in the communication schemes. Inspired by [297, Table 1]. (Continued)

Notation	Description
$Sign(H,\ E_{PrivateKey})$	Creates a signature S of the hash H by encrypting H with $E_{PrivateKey}$, such that decrypting S with $E_{PublicKey}$ returns H, i.e., $AsymDec(S, E_{PublicKey}) = H$.
$[Message]_E$	Represents data that is signed by $E_{PrivateKey}$. It is shorthand for $Message \mid\mid Sign(Hash(Message), E_{PrivateKey})$.
$Create_{Obj}(args)$	A function that creates an object of type *Obj*. Optional arguments $args$ can also be included. The creation details may vary with different values of Obj and $args$.
$Request(Obj)$	Creates a flag for requesting an item or functionality of type Obj. Such flags are used in messages to model the various requests sent between entities in UniSUF (see Section 9.6).
$Success(Obj)$	Creates a flag stating that item of type *Obj* was successfully initiated. Such flags are used in messages to model success statuses sent between entities in UniSUF (see Section 9.6).
$i_1 \mid\mid \ldots \mid\mid i_n$	Represents concatenation of multiple items, more specifically from i_1 to i_n. When the three dots operator $(\ldots)$ notation is used at the end of the concatenation, e.g. $i \mid\mid \ldots$, it indicates that additional but unspecified items are being concatenated after i. Note that an item can be any data.
$(i_1,\ \ldots,\ i_i,\ \ldots) = I$	Items i_1 and up to i_i are extracted from the set of items I. The optional dots at the end signify that more items left in I are ignored.

Table 9.1: Notations used in the communication schemes. Inspired by [297, Table 1]. (Continued)

9.3.4 Update Rounds

Where applicable, cryptographic materials are assigned to individual vehicle identification numbers (VIN), v_{id}, and must be distributed within a specified deadline, t_e (expiration time) [19, 308]. The mapping is secured by appending the v_{id} and t_e to the cryptographic material and signing the resulting data.

We use the pair (v_{id}, t_e) to refer to the software *update rounds* in UniSUF. We choose the term rounds, rather than sessions, to avoid confusion with the term sessions used, e.g., for SSH [309, Sec. 2]. When no v_{id} can be specified, we omit the VIN from our update round notation and use only t_e.

For a given problem and its algorithm, an execution of an algorithm is denoted as an update round execution. We assume that message transmissions include an update round identifier. Therefore, entities can learn about new update rounds and associate each execution of their task sequences with an update round. We denote this execution as an entity's execution of an update round, which is a subsequence of the update round execution. We also assume that all entities have a persistent log of all received messages. Each message is identified by the tuple (r, d), where r is the update round identifier and d is the cryptographic material in the message. All entities drop any message that is already in the persistent log.

9.3.5 Problem Definition

We present our requirements for UniSUF, using the goals derived in 9.1.1.

System-level Requirements 9.3.5.1 to 9.3.5.6 specify UniSUF at the system level. The System-level Requirements 9.3.5.1 to 9.3.5.3 and 9.3.5.5 depend on requirements specified for each UniSUF sub-problem. These requirements are presented in Section 9.6. Namely, one derives the specifications of UniSUF sub-problems by specifying the sub-problems' *set of secrets* (S), the *set of cryptographic materials* $(\mathcal{D})$, the *set of procedures* $(\{\ell_1,\ \ell_2,\ \dots\})$ that handles cryptographic materials, and the ordering constraints on the procedure invocations, which we call the *handling partial order* $(\mathcal{P}(\ell))$.

System-level Requirement 9.3.5.1 (Confidential Secrets)**.** *Let X be an update round execution and S be the* set of secrets *in X, which we specify per sub-problem (see Section 9.6). There is no $s_i \in S$ such that an adversary A can obtain s_i during X.*

System-level Requirements 9.3.5.2 to 9.3.5.4 consider a set of cryptographic materials $\mathcal{D}$, which we specify for each sub-problem in Section 9.6. Each element in $\mathcal{D}$ is a pair; the first element is the cryptographic material itself, and the second element is the material's designated origin entity. As certificates are pre-existing cryptographic materials, we state that the origin entity of each certificate is the root CA.

System-level Requirement 9.3.5.2 requires that the adversary must not manipulate the cryptographic materials in $\mathcal{D}$.

System-level Requirement 9.3.5.2 (Integrity of Cryptographic Materials)**.** *UniSUF only uses cryptographic materials created by their designated origin entity, which we specify in Section 9.6, and is not modified by any other entity.*

System-level Requirement 9.3.5.3 considers procedures that handle cryptographic materials, such as material production, sending, receiving, validation, and software installation. To safeguard the vehicle during the most critical part of the update process, the vehicle enters offline mode [19, 31, 308]. Additionally, the ECUs are normally locked for security reasons, but must be unlocked to install new software. Therefore, we also classify ECU unlocking and vehicle offline mode activation as handling events.

Let X be an execution of update round r, d (documents) a subset of cryptographic materials, and ℓ a name of a procedure that handles d in event $e(r, d, \ell) \in X$. In Section 9.6, we list these procedures per task. We state that $e(r, d, \ell)$ is a handling event in X. Note that $d \subseteq \mathcal{D}$.

System-level Requirement 9.3.5.3 specifies that cryptographic materials coupled with their handling procedures and associated with a specific update round are processed only during that update round; i.e., replays of round unique cryptographic materials between update rounds are not allowed.

System-level Requirement 9.3.5.3 (Inter-Round Uniqueness)**.** *Let $e(r, d, \ell)$ and $e(r', d', \ell')$ be handling events during update round executions X and X', respectively. Suppose $(d, \ell) = (d', \ell')$. It holds that $r = r'$.*

System-level Requirement 9.3.5.4 specifies that cryptographic materials coupled to a specific update round are only processed once per procedure during that update round. In other words, replays of materials in an update round are not allowed.

System-level Requirement 9.3.5.4 (Intra-Round Uniqueness)**.** *Let X be an update round execution and $e(r, d, \ell) \in X$ be a handling event. No event $e'(r, d, \ell)$ exists in X.*

System-level Requirement 9.3.5.5 specifies that procedures are executed in an order that follows UniSUF's specification.

System-level Requirement 9.3.5.5 (Integrity of Handling Events)**.** *Let X be an update round execution. The occurrences of handling events, $e(r, d, \ell) \in X$, must follow a handling partial order $\mathcal{P}(\ell)$, which depends only on ℓ, where $\mathcal{P}(\ell)$ is specified per task (see Section 9.6).*

System-level Requirement 9.3.5.6 prevents non-termination of update rounds, given our system assumptions. The termination is considered timely if all UniSUF entities take the halt task before the update round expires; if not, it is considered late.

System-level Requirement 9.3.5.6 (Termination)**.** *All executions of update rounds must terminate.*

Using the goals specified in Section 9.1.1, we discuss how the requirements satisfy the goals. Firstly, **G1** and **G2** are covered by **Confidential Secrets**, as we ensure that both the software and the cryptography keys are part of the set of secrets S. Note that **G2** is only partially covered because ProVerif [288] can only prove that the adversary is unable to learn the secrets and not that each secret is only available to a certain set of entities. Secondly, **G3** and **G4** are fulfilled by **Integrity of Cryptographic Materials** because the software and other materials produced by the origin entities are never modified throughout the update process. Thirdly, **Inter-Round Uniqueness** and **Intra-Round Uniqueness** together satisfy **G5** because any update round that processes the installation of software cannot be replayed to execute previous versions. **G6** is fulfilled because any VUUP can only exist in a single update round, and such an update round is directly coupled to a vehicle via the VIN (see Section 9.3.1). Fourthly, **G7** is fulfilled by **Integrity of Handling Events**, and lastly, **G8** is achieved by **Termination** since this requires that all executions terminate, legitimate or illegitimately.

9.4 UniSUF Architecture

In this section, we detail the UniSUF architecture and its functionalities.

9.4.1 Cryptographic Materials

As detailed in Table 9.2 and previously mentioned in Section 9.3.3, UniSUF uses different cryptographic materials. The signing process is shown in Figure 9.1.

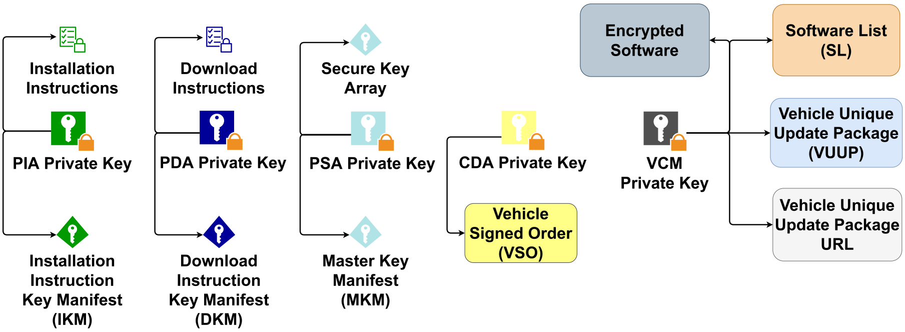

Installation Instructions
PIA Private Key
Installation Instruction Key Manifest (IKM)
Download Instructions
PDA Private Key
Download Instruction Key Manifest (DKM)
Secure Key Array
PSA Private Key
Master Key Manifest (MKM)
CDA Private Key
Vehicle Signed Order (VSO)
Encrypted Software
VCM Private Key
Software List (SL)
Vehicle Unique Update Package (VUUP)
Vehicle Unique Update Package URL

Figure 9.1: Different cryptographic materials in UniSUF are shown, each with its respective key. Figure derived from [31, Fig. 3].

Cryptographic Material	Description
Certificate Package	$PDA_{Cert} \mid\mid PIA_{Cert}$
Download Instructions	$Create_{DownloadInstructions}([SoftwareList]_{VCM})$
Download Instruction Key Manifest (DKM)	$AsymEnc(DKM_{Key}, Vehicle_{PublicKey})$ $\mid\mid DKM_{Policy}$
Installation Instructions	$Create_{InstallationInstructions}($ $[SoftwareList]_{VCM} \mid\mid [SKA]_{PSA}$ $\mid\mid [MKM]_{PSA} \mid\mid$ $\mid\mid PSA_{Cert})$
Installation Instruction Key Manifest (IKM)	$AsymEnc(IKM_{Key}, Vehicle_{PublicKey})$ $\mid\mid IKM_{Policy}$
Master Key Manifest (MKM)	$MKM_{SecurityAccess} \mid\mid MKM_{Software} \mid\mid \ldots$

Continued on next page

Table 9.2: UniSUF cryptographic materials, sorted in alphabetical order.

Cryptographic Material	Description
$MKM_{SecurityAccess}$	$AsymEnc(MKM_{SecurityAccess_{Key}},$ $Vehicle_{PublicKey})$ $\Vert\ MKM_{SecurityAccess_{Policy}}$
$MKM_{Software}$	$AsymEnc(MKM_{Software_{Key}}, Vehicle_{PublicKey})$ $\Vert\ MKM_{Software_{Policy}}$
Secure Key Array (SKA)	$SKA_{SecurityAccess} \Vert SKA_{Software} \Vert \dots$
$SKA_{SecurityAccess}$	$AuthSymEnc($ $SecurityAccess_{Key_1}, MKM_{SecurityAccess_{Key}})$ $\Vert \dots$ $\Vert\ AuthSymEnc($ $SecurityAccess_{Key_n}, MKM_{SecurityAccess_{Key}})$
$SKA_{Software}$	$AuthSymEnc(Software_{Key_1},$ $MKM_{Software_{Key}})$ $\Vert\ \dots$ $\Vert\ AuthSymEnc(Software_{Key_n},$ $MKM_{Software_{Key}})$
Software	$Version \Vert Content$
$Software_{Encased}$	$[SymEnc([Software]_{Supplier},$ $Software_{Key})]_{VCM}$
Vehicle Unique Update Package (VUUP)	$VCM_{Cert} \Vert [VUUP_{Content}]_{VCM}$
$VUUP_{Content}$	$CertificatePackage$ $\Vert\ [SymEnc(DownloadInstructions,$ $DKM_{Key})]_{PDA}$ $\Vert\ [DKM]_{PDA}$ $\Vert\ [SymEnc(InstallationInstructions,$ $IKM_{Key})]_{PIA}$ $\Vert\ [IKM]_{PIA}$

Table 9.2: UniSUF cryptographic materials, sorted in alphabetical order. (Continued)

UniSUF uses Vehicle Unique Update Packages (VUUP) to install vehicle updates. A VUUP is an update package produced by UniSUF for a specific vehicle. It contains all the necessary cryptographic keys, certificates, and instructions for the software update. The internal structure of a VUUP file is shown in Figure 9.2. The VUUP does not contain the actual software files; instead, Download Instructions are included to specify where software files can be downloaded. The specific implementation of Download Instructions is unspecified but can be seen as URLs to the software update files. Note that the Download Instructions is encrypted by a unique session key coupled to its update round. This key is retrieved from the Download Instruction Key Manifest (DKM).

As shown in Figure 9.3, a key manifest consists of a symmetric session key that has been asymmetrically encrypted, accompanied by a policy defining the key's usage (see DKM, IKM, and MKM). Note that the UniSUF term session is equivalent to update rounds (see Section 9.3.4). In Table 9.2, none of the key manifests are signed, to remain consistent with the notation used by Strandberg et al. [31]. However, when the key manifests are transmitted during tasks (see Section 9.6), all key manifests are signed with the certificates according to Figure 9.1.

Additionally, a VUUP includes Installation Instructions encrypted by the session key from the Installation Instruction Key Manifest (IKM). The Installation Instructions contains the diagnostic instructions for installing software, the Master Key Manifest (MKM), and the Secure Key Array (SKA), as shown in Table 9.2. To guide task-specific requirements of Confidential Secrets (see System-level Requirement 9.3.5.1), we define the notations MKM_{Key} and SKA_{Key} to refer to all keys in MKM and SKA respectively.

Thus, the DKM and IKM session keys are packaged into key manifests (i.e., a master key plus a policy). In contrast, the MKM can contain multiple master keys, whereas the DKM and IKM each contain only one [31].

Strandberg [308] defines encased software as software that undergoes a multilayered protection process. Initially, the software is signed by the software supplier to ensure its integrity and authenticity. Next, as part of UniSUF, it is encrypted to ensure confidentiality. Finally, the encrypted software is secured with an additional signature, ensuring that the encrypted package can be validated to avoid initiating the decryption process in case of validation failure. The word encased should not be mixed up with the term encapsulated used by Strandberg et al. [31] to represent materials contained in the VUUP file.

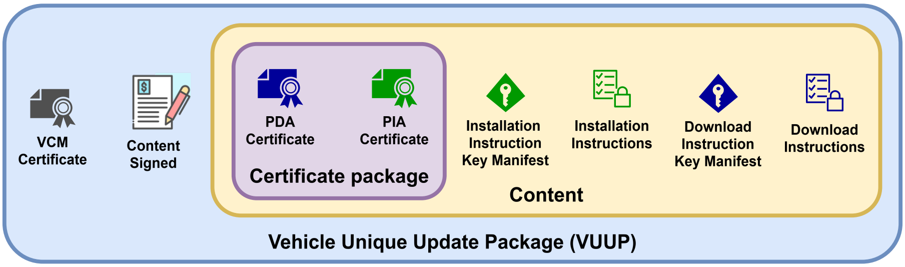

VCM Certificate
Content Signed
PDA Certificate
PIA Certificate
Certificate package
Installation Instruction Key Manifest
Installation Instructions
Download Instruction Key Manifest
Download Instructions
Content
Vehicle Unique Update Package (VUUP)

Figure 9.2: Internal structure of a VUUP file. The blue items are used in the download process, while the green ones are used for the installation process. Note that the VUUP content has been signed by VCM_{Cert}. The figure is derived from [31, Fig. 3].

UniSUF uses the term category to classify master keys based on purpose, such as decrypting software files, or unlocking ECUs to enable update capabilities.

The Vehicle Signed Order (VSO) is a readout per vehicle that contains detailed information about the vehicle, such as the onboard software versions. UniSUF uses this information and the latest available software versions to construct a software list containing the software files and configurations for a specific and vehicle unique software update. Download Instructions and Installation Instructions are then created based on the software list.

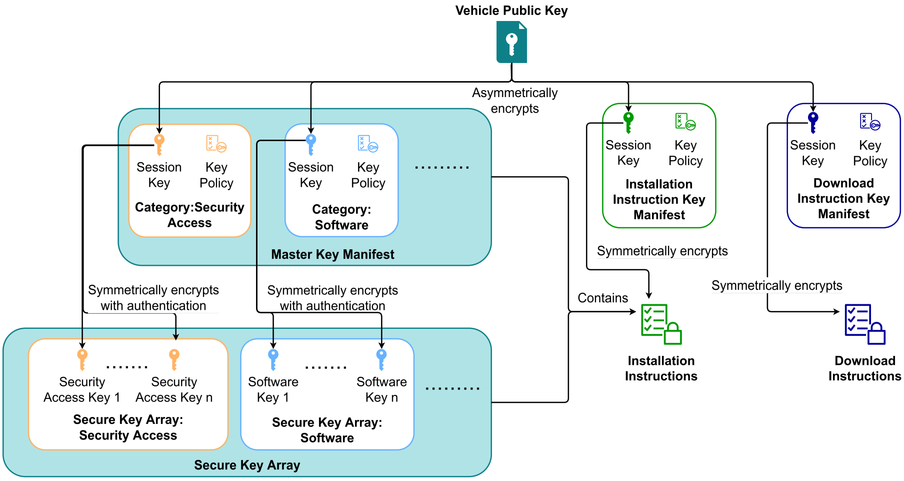


Figure 9.3: Different cryptographic materials in UniSUF and how they are encrypted. Figure derived from [31, Fig. 2].

The software itself is located in external sources and not present in the actual VUUP file. The software files are signed by the software suppliers and associated with version numbers to prevent installations of older software versions [308]. UniSUF validates the supplier signature, further encrypts the software, and appends another signature. Finally, the signed encrypted software is uploaded to the software repository.

9.4.2 System Entities

As shown in Figure 9.4, UniSUF consists of three entities: Producer, Consumer, and the Software Repository [19, Sec. 4]. Additionally, UniSUF interacts with external entities, such as Software Suppliers and ECUs [308]. UniSUF uses redundant entities and interacts with multiple Software Suppliers and vehicles with multiple ECUs [19, 31]. However, for our proof, we consider a simplified system in which each vehicle has exactly one Consumer and one ECU. Additionally, all vehicles communicate with exactly one Producer and one Software Repository, and there exists only one Software Supplier.

9.4.2.1 Software Repository

The UniSUF Software Repository is an entity that represents multiple distributed repositories [31], mainly responsible for software storage, where each software file is associated with a specific download URL. However, in offline cases, the software can also be stored on Network-Attached Storage (NAS) or a USB stick [308].

9.4.2.2 Producer

Strandberg et al. [31] define the Producer as a collection of different sub-entities responsible for producing and securing software update packages. The Producer is also responsible for disseminating software to different storage repositories.

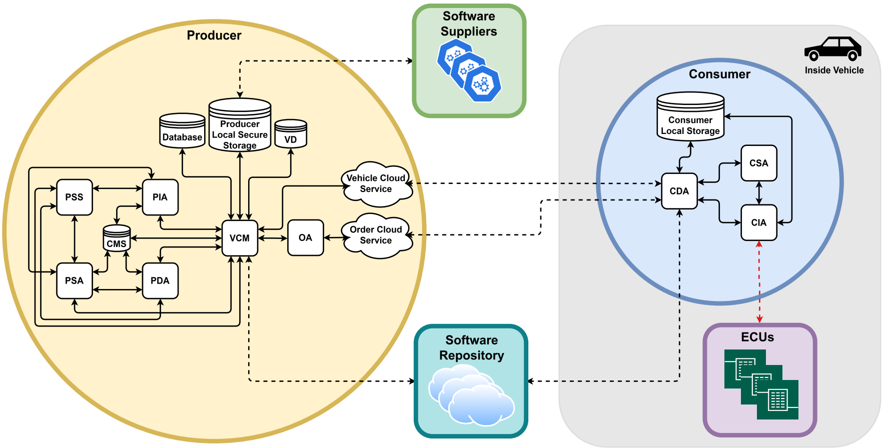


Figure 9.4: Diagram of all the entities and their communication in UniSUF. Dotted arrows denote communication channels between entities from different modules. The black arrows denote secure communications, while the sole red arrow denotes an insecure communication link.

As shown in Figure 9.5, UniSUF has the following 12 producer entities (cf. [31], Table 1).

- **Producer Local Secure Storage** – Stores software files received from software suppliers.
- **Version Control Manager (VCM)** – Coordinates the producer entities and finalizes the creation of the VUUP file for a specific vehicle.
- **Producer Signing Service (PSS)** – Produces signatures on behalf of other entities.
- **Cryptographic Material Storage (CMS)** – Securely stores cryptographic materials, such as keys for decrypting software or unlocking ECUs, as well as certificates.
- **Producer Security Agent (PSA)** – Generates session keys and retrieves additional keys from the CMS, such as keys for unlocking ECUs, performing privileged diagnostic requests, and decrypting software. These additional keys are further encrypted with session keys, which are, in turn, encrypted using a vehicle-specific public certificate.
- **Database** – Store URLs to software files located in software repositories.
- **Order Cloud Service** – Stores the Vehicle Signed Order (VSO) in a queue and URLs to VUUP files.
- **Order Agent (OA)** – Verifies the validity of incoming VSOs and starts the updating process by forwarding the request to the Version Control Manager (VCM).

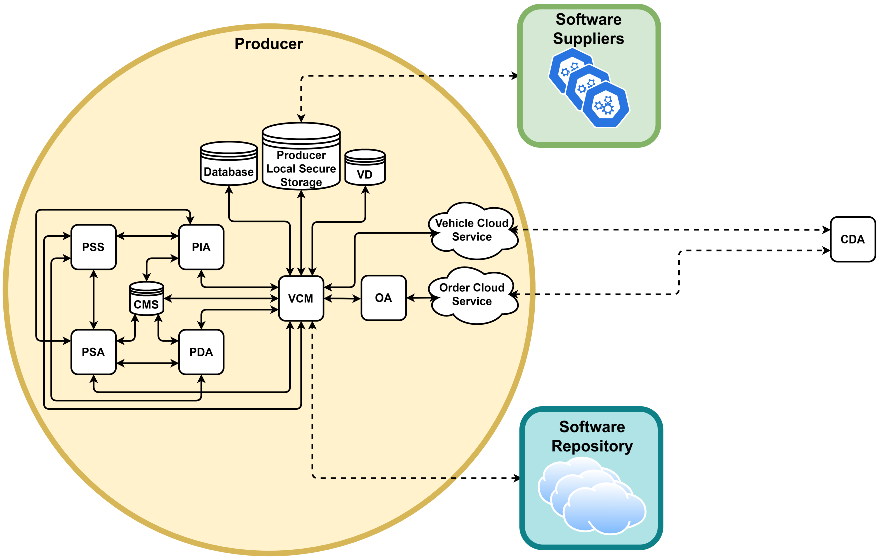


Figure 9.5: The communication flow between the Producer entities. Dotted arrows denote communication channels between a Producer entity and a non-Producer entity.

- **Producer Download Agent (PDA)** – Creates download instructions from the software list received from VCM. Later in the process, the download instructions are encrypted, signed, and sent to VCM.
- **Producer Installation Agent (PIA)** – Creates installation instructions from the software list received from VCM. The installation instructions are bundled with cryptographic material, encrypted, signed, and sent to VCM.
- **VIN Database (VD)** – Stores data about unique vehicles and software versions.
- **Vehicle Cloud Service** – Stores the VUUP files and VCM certificates that can be downloaded by the Consumer via a VUUP URL.

9.4.2.3 Consumer

The Consumer is the Producer's counterpart and is responsible for decapsulating the VUUP and the processing of the download and installation instructions [31, Sec. 4.2]. UniSUF has the following four Consumer entities as shown in Figure 9.6.

- **Consumer Local Storage** – Stores signed VUUP, and signed and encrypted software files.
- **Consumer Download Agent (CDA)** – Executes the download instructions and retrieves software from software repositories.
- **Consumer Security Agent (CSA)** – Provides a trusted execution environment where, e.g., decryption can occur in an isolated and secure space.

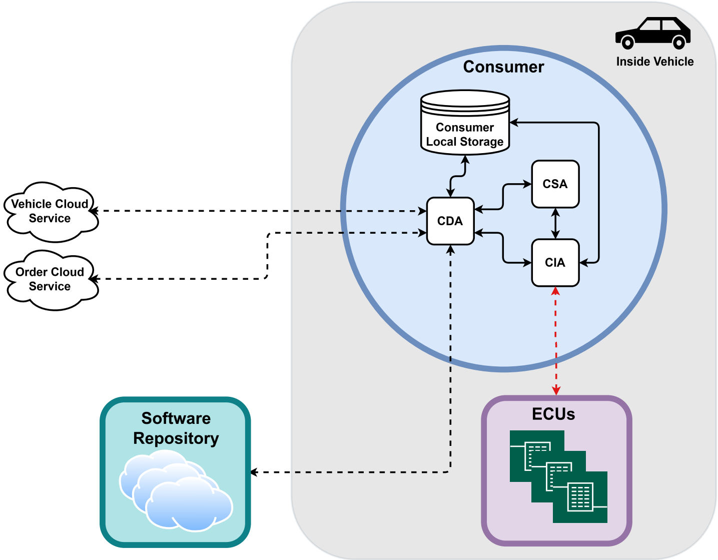


Figure 9.6: The communication flow between the Consumer entities is denoted by the solid arrows. Dotted arrows denote communication between a Consumer and a non-Consumer entity. The red arrow denotes an insecure communication channel. All other channels are secure.

- **Consumer Installation Agent (CIA)** – Executes the installation instructions and then streams the decrypted software to the unlocked ECUs with the help of CSA.

9.4.2.4 Software Suppliers

Software suppliers create and deliver software to the Producer for the installation in different vehicles [31, Sec. 3.3]. The software suppliers also sign their software to provide authenticity. We assume a simplified model that considers a single software supplier supplying software to a single ECU (see Section 9.4.2.5).

9.4.2.5 The Electronic Control Unit

An Electronic Control Unit (ECU) is a vehicle computer responsible for various tasks, from simple signal processing to more advanced functionality, for instance, an infotainment system running various applications. For our simplified model, we assume a system with only one ECU. The ECU is first unlocked and put in programming mode by using security access and a secret key [31, Sec. 4.2], to allow the ECU to receive and install software with Unified Diagnostic Services (UDS) [21, 308].

9.4.2.6 Adversary

As shown in Figure 9.7, the adversary is based on the Dolev-Yao model, assuming an adversary with access to the communication channels.

The adversary is actively present on all communication links in the system. However, most of these links, except the one depicted in red, are secure and reliable communication channels; that is, the adversary cannot interfere, according to Dolev-Yao. The one exception is the link between the Consumer and the ECUs, which is a reliable but not a secure communication channel, where the adversary can potentially read, modify, delete, or insert messages.

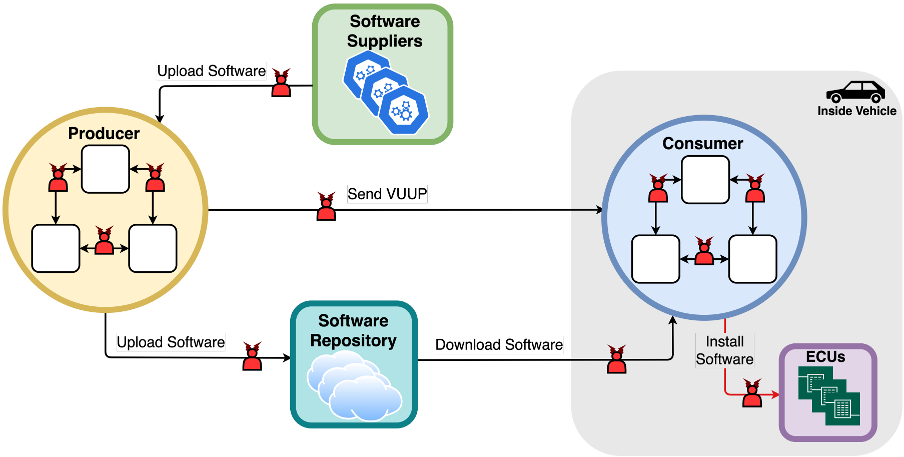


Figure 9.7: The illustration depicts a simplified UniSUF architecture, emphasizing the presence of adversaries represented by the red devils.

9.5 Modelling UniSUF

This section explains the modeling of entities and their respective executions in UniSUF.

9.5.0.1 Execution of System Entities

Each system entity runs a sequence of tasks for a given problem. The entity running the initial user-invoked task is called the *initiator*; illustrated in Figure 9.8. All other entities are listeners (see Figure 9.9) since their first task is the listening task. This task listens for an initiation message specified for each listener. The listening task invokes the next task in sequence and a new listening task, enabling concurrent executions of the task sequence. We divide listeners into two categories: a passive listener that will wait for an initiation message and an active listener that will request an initiation message.

9.5.0.2 Lifecycle of Update Rounds

An update round execution begins when the round identifier is created during the initiation task. Throughout this round, listener entities start executing once their listening task receives a message containing the round identifier.

As shown in Figures 9.10 and 9.11, an entity maintains a context for each update round. This context contains the update round identifier and the

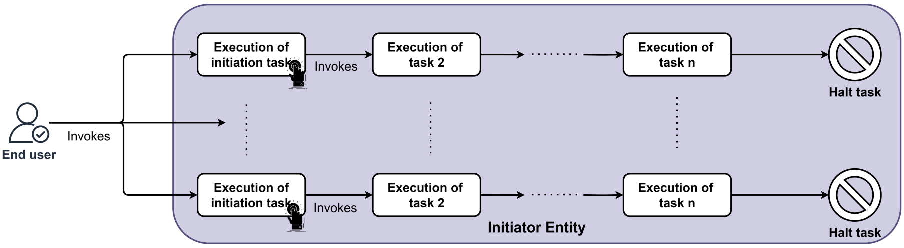


Figure 9.8: The initial user-invoked task begins an execution of the initiator entity's tasks. The initiation tasks are marked with the symbol of a hand pressing a button.

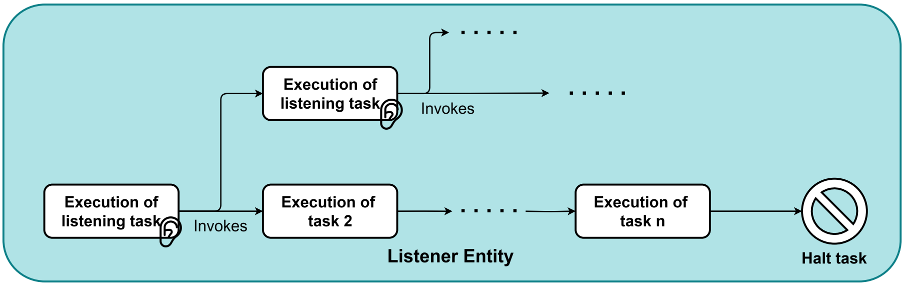


Figure 9.9: The listening task invokes a copy of itself to continue listening on a new execution. It also begins an execution of the listening entity's tasks. The listening tasks are marked with a symbol of an ear.

cryptographic materials (see Section 9.4.1) used throughout the execution of the round. The context is passed along the task sequence and can be updated by each task.

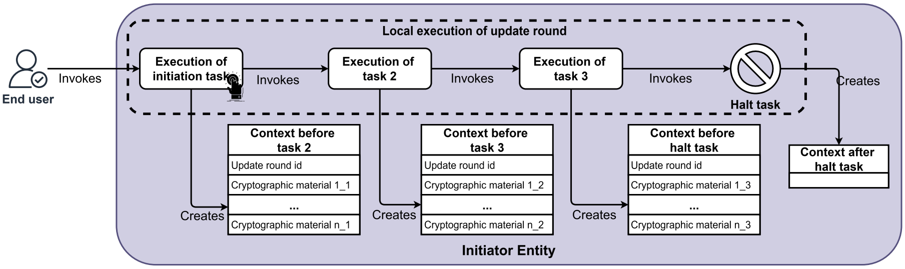


Figure 9.10: The figure illustrates an example of executing an initiator entity's task sequence. For each task, we also show what the context can contain.

Each update round identifier encodes its expiration time and when such expiration occurs, all entities *halt* their local execution of the update round, by removing the context associated with the round and ignoring any further messages related to that round. The update round is terminated once all entities have halted their execution of the update round.

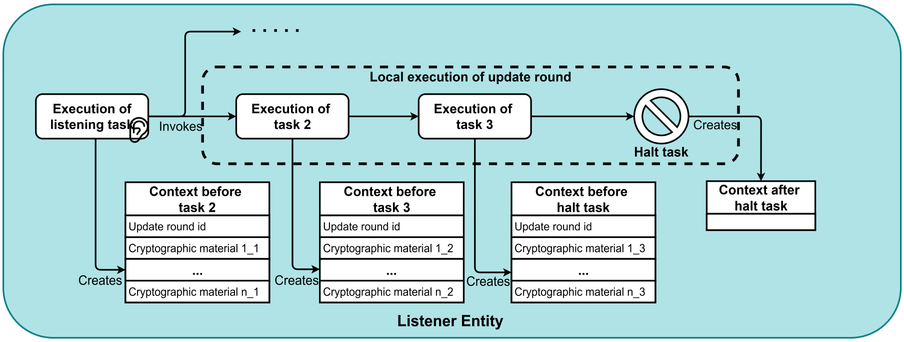


Figure 9.11: The figure illustrates an example of an execution of a listening entity's task sequence. For each task, we also show what the context can contain.

9.5.0.3 Passing Contexts Across Segments of Task Sequences

We divide the main problems into sub-problems, where each sub-problem uses a subset of the system entities. In Figure 9.12, *sub-problem 1* uses the *initiator* and *listener entity 1*. Figure 9.12 shows that an entity's task sequence can be segmented so that each segment belongs to a single sub-problem. For example, *listener entity 1*'s *task 3* and *halt task* form a segment that belongs to *sub-problem 2*.

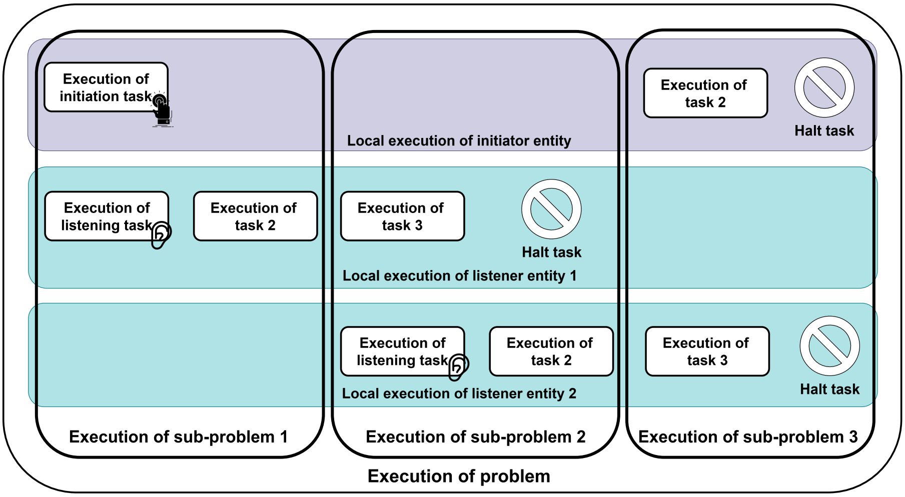

Execution of initiation task
Local execution of initiator entity
Execution of task 2
Halt task
Execution of listening task
Execution of task 2
Execution of task 3
Halt task
Local execution of listener entity 1
Execution of listening task
Execution of task 2
Local execution of listener entity 2
Execution of task 3
Halt task
Execution of sub-problem 1
Execution of sub-problem 2
Execution of sub-problem 3
Execution of problem

Figure 9.12: The execution of an example problem. This problem considers three entities, whereas one is the initiator entity. The problem is divided into three sub-problems.

When an entity starts working on a sub-problem for an update round, it possibly already has an update round context. For instance, if the entity has previously participated in other sub-problems for the same update round, i.e., it has previously run either the initiation task or listening task for the same update round.

Figure 9.13 shows that *listening entity 2* has a context at the start of the sub-problem since it starts on *task 2*, i.e., it has previously run the *listening task*. The other listener, *listener entity 1*, does not start with any context, since it first needs to run its listening task at the start of the sub-problem.

Once a segment's execution finishes, the last context of this segment is passed to the next segment as the starting context. Therefore, we identify which entities have previously executed task segments in the same update round for a given sub-problem. For these entities, we specify the cryptographic materials present in their starting contexts (see Section 9.6).

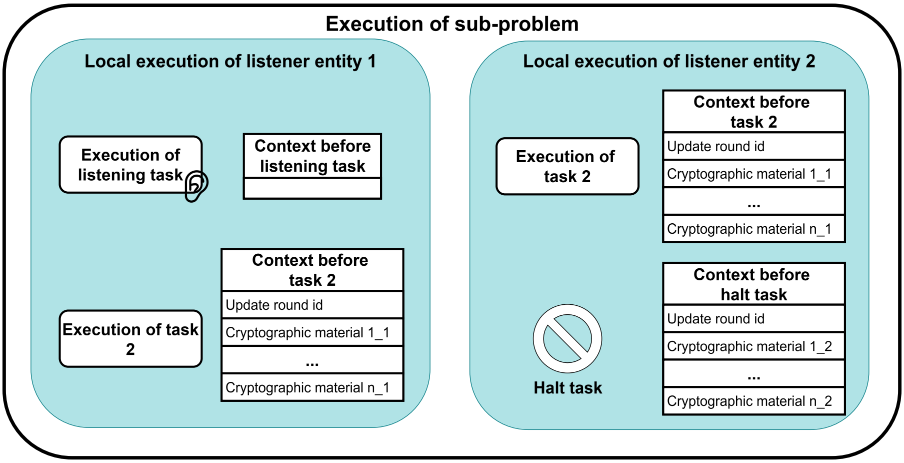

Execution of sub-problem
Local execution of listener entity 1
Execution of listening task
Context before listening task
Context before task 2
Update round id
Cryptographic material 1_1
...
Cryptographic material n_1
Execution of task 2
Local execution of listener entity 2
Context before task 2
Update round id
Cryptographic material 1_1
...
Cryptographic material n_1
Execution of task 2
Context before halt task
Update round id
Cryptographic material 1_2
...
Cryptographic material n_2
Halt task

Figure 9.13: Example of two entities solving a sub-problem together. The figure also shows how each task in the entities is connected to a context.

9.5.0.4 Well-Known Addresses of UniSUF Entities

We assume that all consumer entities, as defined in Section 9.4.2.3, are aware of the specific Vehicle Identification Number (VIN) of the vehicle they occupy. Each vehicle has a pre-stored public vehicle certificate containing metadata, including VIN-related information [308].

Additionally, we assume that all entities in UniSUF know each other's addresses, e.g., through existing protocols, such as DNSSEC, enabling entities to securely obtain IP addresses from domain names [310].

9.6 Sub-Problems

From the UniSUF specifications [31], we analyse two problems: the *software preparation* and the *software update*. The software preparation process (cf. Section 9.6.1) involves safeguarding software received from suppliers, ensuring the software's confidentiality and authenticity for it to be securely incorporated into future vehicle updates. Additionally, the software update is further divided into the *encapsulation* and the *decapsulation* stages (cf. Sections 9.6.2 and 9.6.3) emphasizing securely updating vehicular software and configurations. Strandberg et al. [31] specify an additional stage after decapsulation: the post-state. The post-state encompasses installation reports and logs, potentially affecting upcoming software updates. However, we do not consider the post-state in our simplified model.

Consequently, the main tasks in UniSUF are dissected into sub-problems, based on the steps provided by [31], where each sub-problem, has a description, a diagram depicting its algorithm, a communication scheme, and assumptions and requirements.

9.6.1 Preparation

An overview of the entities involved in the preparation stage and their communication links can be seen in Figure 9.14.

In the preparation stage, software supplier files are processed before being used for software updates [31]. We have divided this stage into two sub-problems: the *Secure Software Files* and the *Upload Software Files*. The focus of the first sub-problem is the encrypting and signing of the software, whereas the second sub-problem handles the software upload and the creation of software URLs.

The preparation stage manages both software files applicable to multiple vehicles and unique files for specific vehicles [308]. In our model, we assume the case when software is being prepared for multiple vehicles and we additionally assume that the update round is solely identified by the expiration time t_e (see Section 9.3.4).

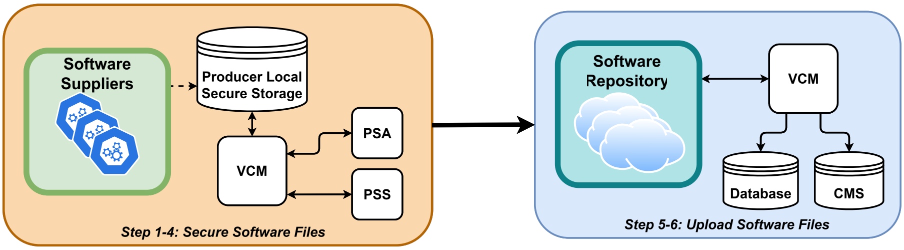

Software Suppliers
Producer Local Secure Storage
VCM
PSA
PSS
Step 1-4: Secure Software Files
Software Repository
VCM
Database
CMS
Step 5-6: Upload Software Files

Figure 9.14: An overview derived from [31, Sec. 3.3] for the communication between all entities involved in the preparation stage further divided into two sub-problems.

9.6.1.1 Step 1–4: Secure Software Files

The initial phase of the update process focuses on securing software files (see Figure 9.15 and Table 9.3). The software supplier signs the software files before sending them to local storage on the producer side. VCM validates the signature and encrypts the software using a symmetric key obtained from PSA; this key is referred to as $Software_{Key}$. The encrypted software is then signed with the VCM's certificate to finalize the $Software_{Encapsualted}$ assembly.

$$\begin{aligned} S =& \{Software,\ Software_{Key},\ Supplier_{PrivateKey},\ VCM_{PrivateKey}\} \\ \mathcal{D} =& \{(Software,\ Supplier),\ (Software_{Key},\ PSA),\ (SoftwareHash,\ VCM), \\ & (SignedSoftwareHash,\ PSS)\} \end{aligned}$$

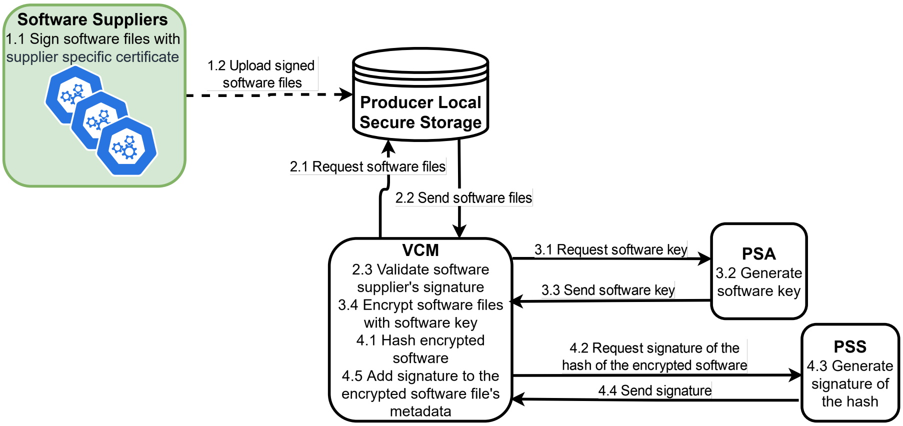

Software Suppliers
1.1 Sign software files with supplier specific certificate
1.2 Upload signed software files
Producer Local Secure Storage
2.1 Request software files
2.2 Send software files
VCM
2.3 Validate software supplier's signature
3.4 Encrypt software files with software key
4.1 Hash encrypted software
4.5 Add signature to the encrypted software file's metadata
3.1 Request software key
3.3 Send software key
PSA
3.2 Generate software key
4.2 Request signature of the hash of the encrypted software
4.4 Send signature
PSS
4.3 Generate signature of the hash

Figure 9.15: Diagram of the sub-problem *Secure Software Files*.

To derive the specific sub-problem requirements from the system requirements (see Section 9.3.5), we define S, $\mathcal{D}$ and the partial order $\mathcal{P}(\ell) = \ell_i < \ell_{i+1}$ for $1 \leq i \leq 4$ with the following labels:

ℓ_1: The software suppliers upload the software to Producer Local Storage.

ℓ_2: PSA generates the software key.

ℓ_3: VCM generates a hash of the software.

ℓ_4: PSS generates a signature for the software.

ℓ_5: VCM assembles $Software_{Encased}$.

9.6.1.2 Step 5–6: Upload Software Files

Once the software file has been encased, it is uploaded into a software repository. The repository then generates a URL to the uploaded encased software and returns this URL to VCM (see steps 5.1 and 5.2 in Figure 9.16 and Table 9.4). Finally, the VCM stores the software URL in the VIN database and stores the $Software_{Key}$ in CMS; used for the encryption of software files.

Note that VCM has a starting context because its initiation task is in a previous sub-problem (see Section 9.6.1.1). VCM's starting context contains $Software_{Encased}$ and $Software_{Key}$.

$$\begin{aligned}
S =& \{Software,\ Software_{Key},\ Supplier_{PrivateKey},\ VCM_{PrivateKey}\}\\
\mathcal{D} =& \{(Software_{Encased},\ VCM),\ (Software_{Key},\ PSA),\\
& (Software_{URL},\ SoftwareRepository)\}
\end{aligned}$$

To derive the specific sub-problem requirements from the system requirements (see Section 9.3.5), we define S, $\mathcal{D}$ and the partial order $\mathcal{P}(\ell) = \ell_i < \ell_{i+1}$ for $1 \leq i \leq 3$ with the following labels:

(1.2)	$Supplier \rightarrow ProducerLocalStorage$	$[Software]_{Supplier}$
(2.1)	$VCM \rightarrow ProducerLocalStorage$	$Request(Software)$
(2.2)	$ProducerLocalStorage \rightarrow VCM$	$[Software]_{Supplier}$
(3.1)	$VCM \rightarrow PSA$	$Request(Software_{Key})$
(3.2)	$PSA \rightarrow VCM$	$Software_{Key}$
(4.2)	$VCM \rightarrow PSS$	$SoftwareHash$ $SoftwareHash = Hash($ $SymEnc([Software]_{Supplier},$ $Software_{Key}))$
(4.4)	$PSS \rightarrow VCM$	$SignedSoftwareHash$ $SignedSoftwareHash =$ $Sign(SoftwareHash,$ $VCM_{PrivateKey})$

Table 9.3: Communication scheme for the sub-problem *Secure Software Files*. The dashed line represents parallel processes.

(5.1)	$VCM \rightarrow SoftwareRepository$	$Software_{Encased}$
(5.2)	$SoftwareRepository \rightarrow VCM$	$Software_{URL}$
(5.3)	$VCM \rightarrow VINDatabase$	$Software_{URL}$
(5.4)	$VINDatabase \rightarrow VCM$	$Success(Software_{URL})$
(6.1)	$VCM \rightarrow CMS$	$Software_{Key}$
(6.2)	$CMS \rightarrow VCM$	$Success(Software_{Key})$

Table 9.4: Communication scheme for the sub-problem *Upload Software Files*.

ℓ_1: Software Repository receives the $Software_{Encased}$.

ℓ_2: VIN Database stores the $Software_{URL}$.

ℓ_3: CMS stores the $Software_{key}$.

ℓ_4: VCM receives status of $Software_{key}$ being stored in CMS.

9.6.2 Encapsulation

The encapsulation stage starts when the Order Agent has received an order request from CDA, whereafter producer entities work collaboratively to produce a VUUP [31, Sec. 4.1]. Figure 9.17 shows a high-level flow diagram of the different sub-problems involved in the encapsulation stage. In steps 1–2, the encapsulation stage starts by producing a VSO, which is later processed to a Software List in step 3. Furthermore, in steps 4–7, the Software List is sent to PDA, PIA, and PSA to generate necessary materials, for instance, download and installation instructions. Instructions and other materials are included in a VUUP file (step 8), whereafter the encapsulation stage finalizes by notifying the consumer that updates are available (cf. steps 9–11).

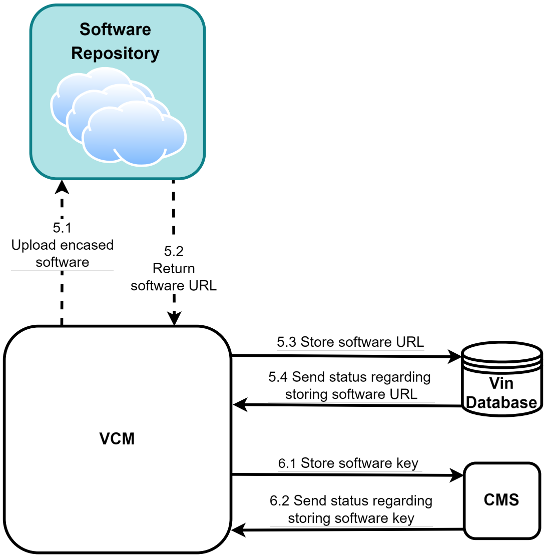

Software Repository
5.1 Upload encased software
5.2 Return software URL
VCM
5.3 Store software URL
5.4 Send status regarding storing software URL
Vin Database
6.1 Store software key
6.2 Send status regarding storing software key
CMS

Figure 9.16: Diagram of the sub-problem *Upload Software Files*.

9.6.2.1 Step 1–2: Order Initiation

The encapsulation process begins when the CDA requests an update by sending a signed order (denoted as VSO) to the Order Cloud Service (see steps 1.1–1.3 in Figure 9.18). In step 1.4, Order Cloud Service stores the order in a queue, whereafter the Order Agent attempts to fetch an order from this queue. If an order is available, Order Cloud Service sends a VSO to Order Agent, which verifies the signature and initiates VCM using the VSO (steps 2.1–2.4).

(1.3)	$CDA \rightarrow OrderCloudService$	$[VSO]_{Consumer}$
(2.1)	$OrderAgent \rightarrow OrderCloudService$	$Request(VSO)$
(2.2)	$OrderCloudService \rightarrow OrderAgent$	$[VSO]_{Consumer}$
(2.4)	$OrderAgent \rightarrow VCM$	$[VSO]_{Consumer}$

Table 9.5: Communication scheme for the sub-problem *Order Initiation*.

To derive the specific sub-problem requirements from the system requirements (see Section 9.3.5), we define $S = \{CDA_{PrivateKey}\}$, $\mathcal{D} = \{(VSO,\ CDA)\}$, and the partial order $\mathcal{P}(\ell) = \ell_i < \ell_{i+1}$ for $1 \leq i \leq 4$ with the following labels:

ℓ_1: CDA generates a signed VSO.

ℓ_2: Order Cloud Service stores the signed VSO in its queue.

ℓ_3: Order Agent pulls signed VSO from Order Cloud Service.

ℓ_4: Order Agent initiates VCM with the signed VCM.

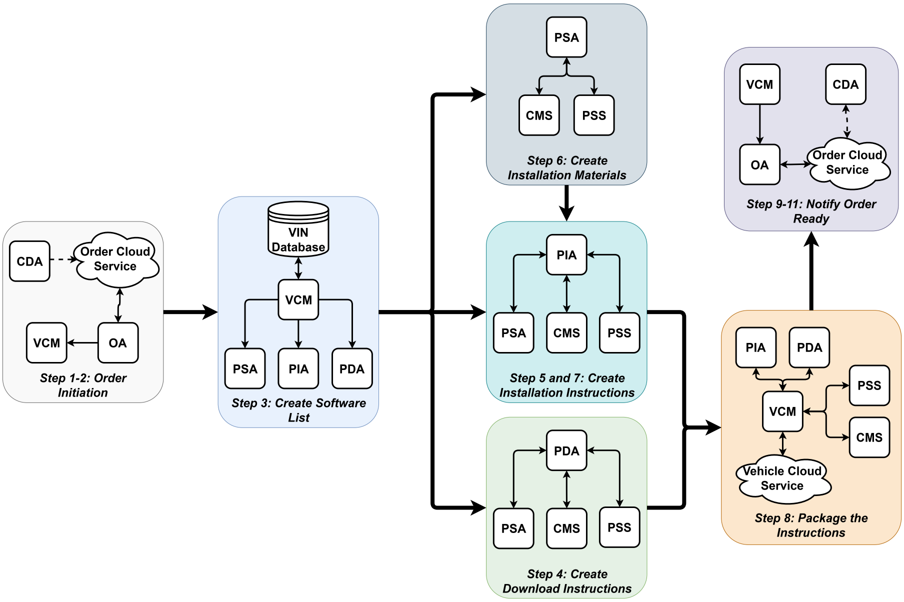

CDA
Order Cloud Service
VCM
OA
Step 1-2: Order Initiation
VIN Database
VCM
PSA
PIA
PDA
Step 3: Create Software List
PSA
CMS
PSS
Step 6: Create Installation Materials
PIA
PSA
CMS
PSS
Step 5 and 7: Create Installation Instructions
PDA
PSA
CMS
PSS
Step 4: Create Download Instructions
VCM
CDA
OA
Order Cloud Service
Step 9-11: Notify Order Ready
PIA
PDA
PSS
VCM
CMS
Vehicle Cloud Service
Step 8: Package the Instructions

Figure 9.17: Overview of communications between all entities that are involved in the encapsulation stage.

ℓ_5: VCM sends status on initialisation.

9.6.2.2 Step 3: Create Software List

Given a VSO, the VCM will produce a software list (see step 3.1–3.5 in Figure 9.19). The software list contains the software to be installed in the vehicle. After creating the software list, it is transmitted to the PIA, PSA, and PDA for further processing (step 3.6). The software list is used to generate the download and installation instructions.

The entity VCM has a starting context because its listening task has been included in the previous sub-problem *Order Initiation* (see Section 9.6.2.1). The starting context of VCM contains the signed VSO.

(3.2)	$VCM \rightarrow VINDatabase$	VIN
		$(VIN, ...) = VSO$
(3.3)	$VINDatabase \rightarrow VCM$	$VIN_{Data} \parallel Software_{Versions}$
(3.6)	$VCM \rightarrow PDA$	$[SoftwareList]_{VCM}$
	$VCM \rightarrow PSA$	$[SoftwareList]_{VCM}$
	$VCM \rightarrow PIA$	$[SoftwareList]_{VCM}$
		$SoftwareList = Create_{SoftwareList}($
		$VIN_{Data} \parallel Software_{Versions})$

Table 9.6: Communication scheme for the sub-problem *Create Software List*.

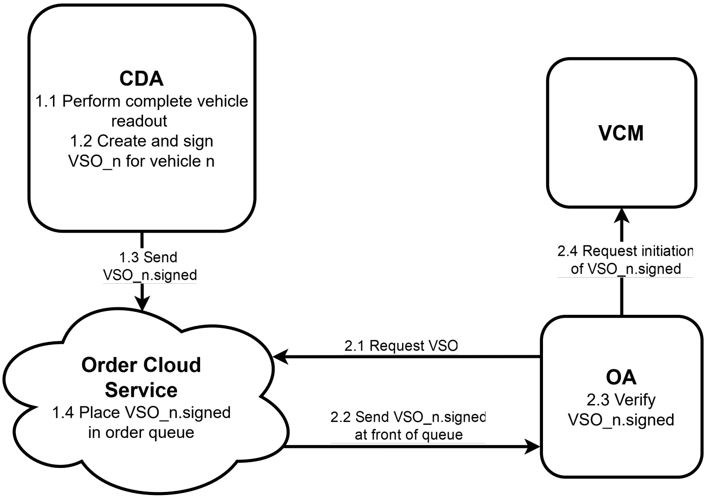

CDA
1.1 Perform complete vehicle readout
1.2 Create and sign VSO_n for vehicle n
VCM
1.3 Send VSO_n.signed
2.4 Request initiation of VSO_n.signed
2.1 Request VSO
Order Cloud Service
1.4 Place VSO_n.signed in order queue
OA
2.3 Verify VSO_n.signed
2.2 Send VSO_n.signed at front of queue

Figure 9.18: Diagram of the sub-problem *Order Initiation.*

$$\begin{aligned}S =&\{CDA_{PrivateKey},\ VCM_{PrivateKey}\}\\ \mathcal{D} =&\{(VSO,\ CDA),\ (SoftwareList,\ VCM),\ (VIN_{Data},\ VINDatabase),\\ &(Software_{Versions},\ VINDatabase)\}\end{aligned}$$

To derive the specific sub-problem requirements from the system requirements (see Section 9.3.5), we define S and $\mathcal{D}$ as seen above, and the partial order $P(\ell) = \{\ell_1 < \ell_2,\ \ell_2 < \ell_3,\ \ell_2 < \ell_4,\ \ell_2 < \ell_5,\ \ell_3 < \ell_6,\ \ell_4 < \ell_6,\ \ell_5 < \ell_6\}$. In other words, ℓ_1 happens before ℓ_2 happens, which then precedes ℓ_3, ℓ_4, and ℓ_5 occurring in parallel, and finally ℓ_6 happens last.

ℓ_1: VCM obtains the most up-to-date software versions and vehicle data from VIN Database.

ℓ_2: VCM creates a signed Software List.

ℓ_3: VCM sends the signed Software List to PDA.

ℓ_4: VCM sends the signed Software List to PIA.

ℓ_5: VCM sends the signed Software List to PSA.

ℓ_6: VCM receives status of Software List being sent to PDA, PIA and PSA.

9.6.2.3 Step 4: Create Download Instructions

Based on the Software List received from the VCM in the previous sub-problem *Create Software List* (see Section 9.6.2.2), PDA creates the Download Instructions (see steps 4.1–4.2). The Download Instructions is encrypted using a generated session key. The session key is then used to produce the key

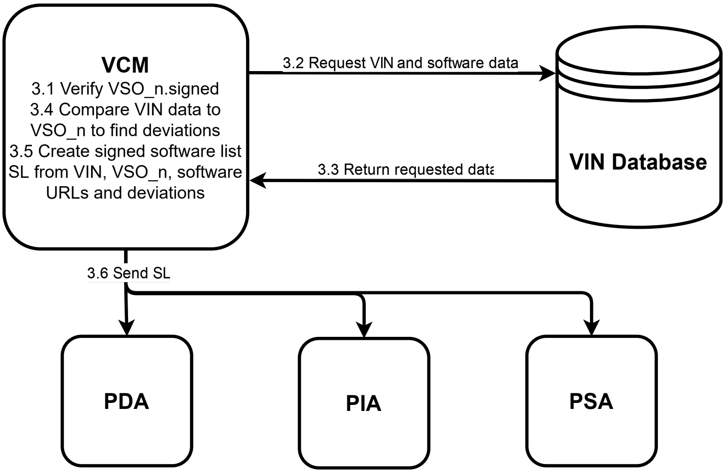

VCM
3.1 Verify VSO_n.signed
3.4 Compare VIN data to VSO_n to find deviations
3.5 Create signed software list SL from VIN, VSO_n, software URLs and deviations
3.2 Request VIN and software data
3.3 Return requested data
VIN Database
3.6 Send SL
PDA
PIA
PSA

Figure 9.19: Diagram of the sub-problem *Create Software List.*

manifest DKM (steps 4.3–4.10). This sub-problem finishes by signing the Download Instructions and DKM (steps 4.12–4.16).

The entities PDA and PSA have starting contexts because their listening tasks have been included in a previous sub-problem (see sub-problem *Create Software List*). The starting context of PDA contains the signed Software List and VCM_{Cert}.

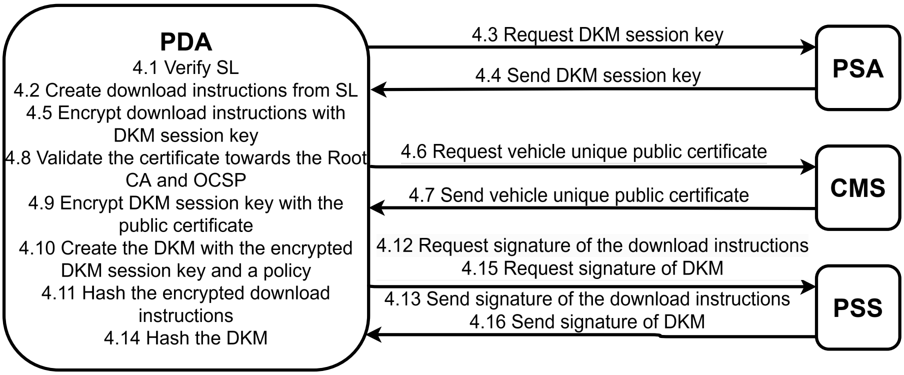

PDA
4.1 Verify SL
4.2 Create download instructions from SL
4.5 Encrypt download instructions with DKM session key
4.8 Validate the certificate towards the Root CA and OCSP
4.9 Encrypt DKM session key with the public certificate
4.10 Create the DKM with the encrypted DKM session key and a policy
4.11 Hash the encrypted download instructions
4.14 Hash the DKM
4.3 Request DKM session key
4.4 Send DKM session key
PSA
4.6 Request vehicle unique public certificate
4.7 Send vehicle unique public certificate
CMS
4.12 Request signature of the download instructions
4.15 Request signature of DKM
4.13 Send signature of the download instructions
4.16 Send signature of DKM
PSS

Figure 9.20: Diagram of the sub-problem *Create Download Instructions.*

$$\begin{aligned} S =&\{SoftwareList,\ DownloadInstructions,\ DKM_{Key},\ Root_{PrivateKey},\\ &PDA_{PrivateKey}, VCM_{PrivateKey}\}\\ \mathcal{D} =&\{(SoftwareList,\ VCM),\ (DownloadInstructions,\ PDA),\\ &(DKM_{Key},\ PSA),\ (Vehicle_{Cert},\ Root),\ (PDA_{Cert},\ Root),\\ &(DKM,\ PDA)\}\end{aligned}$$

To derive the specific sub-problem requirements from the system requirements (see Section 9.3.5), we define the sets S and $\mathcal{D}$ as seen above, and the

(4.3)	$PDA \rightarrow PSA$	$Request(DKM_{Key})$
(4.4)	$PSA \rightarrow PDA$	DKM_{Key}
(4.6)	$PDA \rightarrow CMS$	$Request(Vehicle_{Cert})$
(4.7)	$CMS \rightarrow PDA$	$Vehicle_{Cert}$
(4.12)	$PDA \rightarrow PSS$	$Hash($ $SymEnc($ $DownloadInstructions,$ $DKM_{Key}))$
(4.13)	$PSS \rightarrow PDA$	$Sign($ $Hash($ $SymEnc($ $DownloadInstructions,$ $DKM_{Key})),$ $PDA_{PrivateKey})$
(4.15)	$PDA \rightarrow PSS$	$Hash(DKM)$
(4.16)	$PSS \rightarrow PDA$	$Sign(Hash(DKM),\ PDA_{PrivateKey})$

Table 9.7: Communication scheme for the sub-problem *Create Download Instructions*.

partial order $\mathcal{P}(\ell) = \ell_i < \ell_{i+1}$ for $1 \leq i \leq 10$ with the following labels:

ℓ_1: PDA verifies the Software List

ℓ_2: PDA creates the Download Instructions

ℓ_3: PSA generates DKM_{Key}

ℓ_4: PSA sends DKM_{Key} to PDA

ℓ_5: CMS sends $Vehicle_{Cert}$ to PDA

ℓ_6: PDA generates DKM

ℓ_7: PSS generates a signature for the Download Instructions

ℓ_8: PSS sends the signed encrypted Download Instructions to PDA

ℓ_9: PSS generates signature for DKM

ℓ_{10}: PSS sends the signed encrypted DKM to PDA

9.6.2.4 Step 6: Generate Installation Materials

In parallel to the creation of Download Instructions and Installation Instructions (see Sections 9.6.2.3 and 9.6.2.5), the PSA generates and secures cryptographic materials necessary for the installation of software updates, e.g., SKA and MKM. For instance, ECU keys for the unlocking of ECUs, to gain extended privileges.

The entities PSA, PSS, and CMS have starting contexts because their listening tasks have been included in previous sub-problems (see sub-problem *Create Download Instructions*). PSA's starting context contains the signed Software List and VCM_{Cert} and CMS's starting context contains the $Vehicle_{Cert}$.

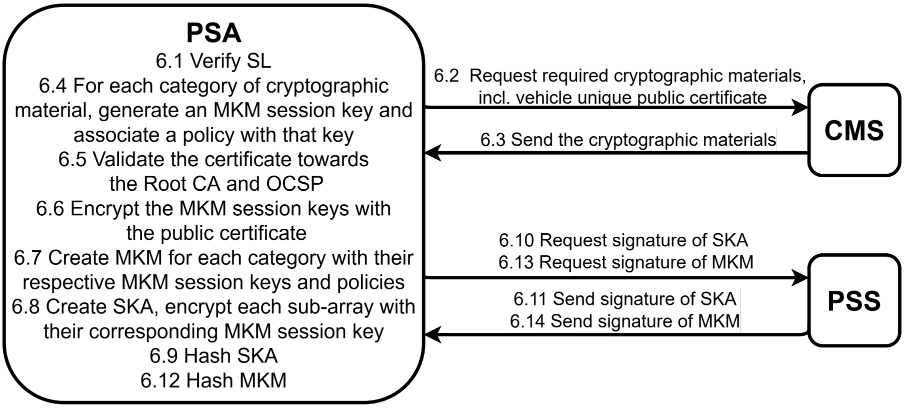

PSA
6.1 Verify SL
6.4 For each category of cryptographic material, generate an MKM session key and associate a policy with that key
6.5 Validate the certificate towards the Root CA and OCSP
6.6 Encrypt the MKM session keys with the public certificate
6.7 Create MKM for each category with their respective MKM session keys and policies
6.8 Create SKA, encrypt each sub-array with their corresponding MKM session key
6.9 Hash SKA
6.12 Hash MKM
6.2 Request required cryptographic materials, incl. vehicle unique public certificate
6.3 Send the cryptographic materials
CMS
6.10 Request signature of SKA
6.13 Request signature of MKM
6.11 Send signature of SKA
6.14 Send signature of MKM
PSS

Figure 9.21: Diagram of the sub-problem *Generate Installation Materials.*

(6.2)	$PSA \rightarrow CMS$	$SoftwareList$
(6.3)	$CMS \rightarrow PSA$	$Vehicle_{Cert} \parallel SKA_{Key}$
(6.10)	$PSA \rightarrow PSS$	$Hash(SKA)$
(6.11)	$PSS \rightarrow PSA$	$Sign(Hash(SKA), PSA_{PrivateKey})$
(6.13)	$PSA \rightarrow PSS$	$Hash(MKM)$
(6.14)	$PSS \rightarrow PSA$	$Sign(Hash(MKM), PSA_{PrivateKey})$

Table 9.8: Communication scheme for the sub-problem *Generate Installation Materials.*

$$\begin{aligned} S =& \{SoftwareList,\ MKM_{Key},\ SKA_{Key},\ \ Root_{PrivateKey},\ PSA_{PrivateKey},\\ & VCM_{PrivateKey}\} \\ \mathcal{D} =& \{(SoftwareList,\ VCM),\ (Vehicle_{Cert},\ Root),\ (PSA_{Cert},\ Root),\\ & (MKM,\ PSA),\ (SKA,\ PSA)\} \end{aligned}$$

To derive the specific sub-problem requirements from the system requirements (see Section 9.3.5), we define the sets S and $\mathcal{D}$ as seen above, and the partial order $\mathcal{P}(\ell) = \ell_i < \ell_{i+1}$ for $1 \leq i \leq 9$ with the following labels:

ℓ_1: PSA verifies the Software List

ℓ_2: CMS sends cryptographic material to PSA

ℓ_3: PSA generates $MKM_{ECU_{Key}}$ and $MKM_{SW_{Key}}$

ℓ_4: PSA generates MKM

ℓ_5: PSA generates SKA

ℓ_6: PSS generates a signature for the SKA

ℓ_7: PSS sends the signed SKA to PSA

ℓ_8: PSS generates signature for MKM

ℓ_9: PSS sends the signed MKM to PSA

9.6.2.5 Step 5 and 7: Create Installation Instructions

The PIA receives Software List from VCM and creates Installation Instructions based on the Software List (see steps 5.1–5.2 in Figure 9.22). Similarly to sub-problem *Create Download Installation*, the Installation Instructions are encrypted with a generated session key, which gives rise to the IKM (steps 7.1–7.11). The Installation Instructions is also appended with the materials generated in the sub-problem *Generate Installation Materials* (steps 7.1–7.3).

The entities PIA, PSA, PSS, and CMS have starting contexts because their listening tasks are included in previous sub-problems (see sub-problems *Create Software List* and *Create Download Instructions*). PIA's starting context contains the signed Software List and VCM_{Cert}, PSA's starting context contains the signed SKA and signed MKM, and CMS's starting context contains the $Vehicle_{Cert}$.

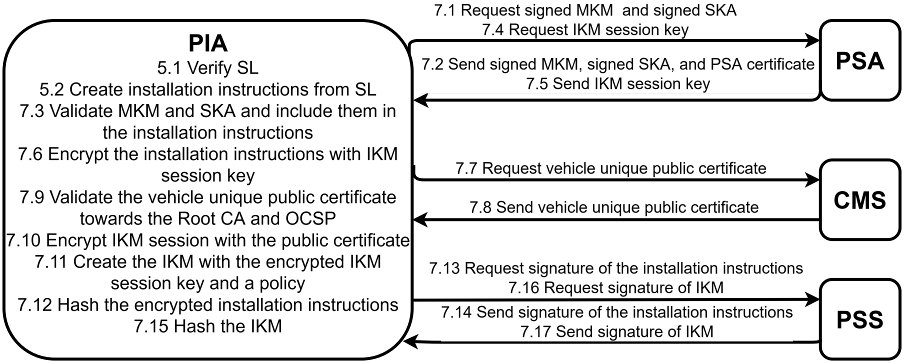

PIA
5.1 Verify SL
5.2 Create installation instructions from SL
7.3 Validate MKM and SKA and include them in the installation instructions
7.6 Encrypt the installation instructions with IKM session key
7.9 Validate the vehicle unique public certificate towards the Root CA and OCSP
7.10 Encrypt IKM session with the public certificate
7.11 Create the IKM with the encrypted IKM session key and a policy
7.12 Hash the encrypted installation instructions
7.15 Hash the IKM
7.1 Request signed MKM and signed SKA
7.4 Request IKM session key
7.2 Send signed MKM, signed SKA, and PSA certificate
7.5 Send IKM session key
PSA
7.7 Request vehicle unique public certificate
7.8 Send vehicle unique public certificate
CMS
7.13 Request signature of the installation instructions
7.16 Request signature of IKM
7.14 Send signature of the installation instructions
7.17 Send signature of IKM
PSS

Figure 9.22: Diagram of the sub-problem *Create Installation Instructions.*

$$
\begin{aligned}
S = &\{SoftwareList,\ InstallationInstructions,\ IKM_{Key},\ Root_{PrivateKey},\\
&PIA_{PrivateKey},\ VCM_{PrivateKey}\}\\
\mathcal{D} = &\{(SoftwareList,\ VCM),\ (InstallationInstructions,\ PIA),\\
&(IKM_{Key},\ PSA),\ (Vehicle_{Cert},\ Root),\ (PIA_{Cert},\ Root),\\
&(MKM,\ PSA),\ (SKA,\ PSA)\}
\end{aligned}
$$

To derive the specific sub-problem requirements from the system requirements (see Section 9.3.5), we define the sets S and $\mathcal{D}$ as seen above, and the partial order $\mathcal{P}(\ell) = \ell_i < \ell_{i+1}$ for $1 \leq i \leq 11$ with the following labels:

ℓ_1: PIA verifies the Software List

ℓ_2: PIA creates the Installation Instructions

ℓ_3: PSA sends $[MKM]_{PSA}$, and $[SKA]_{PSA}$, and PSA_{Cert}

ℓ_4: PSA generates IKM_{Key}

ℓ_5: PSA sends IKM_{Key} to PIA

(7.1)	$PIA \rightarrow PSA$	$Request(MKM) \parallel Request(SKA)$
(7.2)	$PSA \rightarrow PIA$	$[MKM]_{PSA} \parallel [SKA]_{PSA} \parallel PSA_{Cert}$
(7.4)	$PIA \rightarrow PSA$	$Request(IKM_{Key})$
(7.5)	$PSA \rightarrow PIA$	IKM_{Key}
(7.7)	$PIA \rightarrow CMS$	$Request(Vehicle_{Cert})$
(7.8)	$CMS \rightarrow PIA$	$Vehicle_{Cert}$
(7.13)	$PIA \rightarrow PSS$	$Hash($ $\quad SymEnc($ $\quad\quad InstallationInstructions,$ $\quad\quad IKM_{Key}))$
(7.14)	$PSS \rightarrow PIA$	$Sign($ $\quad Hash($ $\quad\quad SymEnc($ $\quad\quad\quad InstallationInstructions,$ $\quad\quad\quad IKM_{Key})),$ $\quad PIA_{PrivateKey})$
(7.16)	$PIA \rightarrow PSS$	$Hash(IKM)$
(7.17)	$PSS \rightarrow PIA$	$Sign(Hash(IKM), PIA_{PrivateKey})$

Table 9.9: Communication scheme for the sub-problem *Create Installation Instructions.*

ℓ_6: CMS sends $Vehicle_{Cert}$ to PIA

ℓ_7: PIA generates IKM

ℓ_8: PSS generates signature for the Installation Instructions

ℓ_9: PSS sends the signed encrypted Installation Instructions to PIA

ℓ_{10}: PSS generates signature for IKM

ℓ_{11}: PSS sends signed encrypted IKM to PIA

9.6.2.6 Step 8: Package the Instructions

Once the required materials for software update have been created, the materials need to be inserted into the VUUP file. The procedure starts with PDA and PIA sending data to VCM (see steps 8.1–8.2 in Figure 9.23). Before data is packaged into VUUP content, signatures are validated to ensure authenticity (steps 8.3–8.8). The VUUP file is signed by the PSS and later uploaded to the cloud (steps 8.9–8.14).

The entities PDA, PIA, VCM, CMS, and PSS have a starting context because their listening tasks are included in previous sub-problems *Order Initiation* (see Sections 9.6.2.2 to 9.6.2.5). PDA's starting context contains the encrypted and signed Download Instructions and the signed DKM, and PIA's starting context contains the encrypted and signed Installation Instructions and the signed IKM.

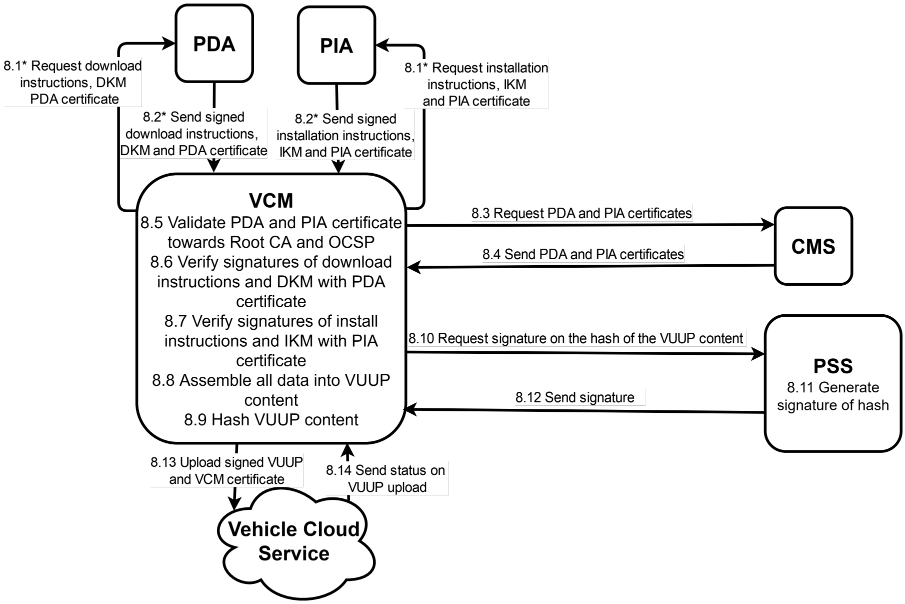

PDA
PIA
8.1* Request download instructions, DKM PDA certificate
8.1* Request installation instructions, IKM and PIA certificate
8.2* Send signed download instructions, DKM and PDA certificate
8.2* Send signed installation instructions, IKM and PIA certificate
VCM
8.5 Validate PDA and PIA certificate towards Root CA and OCSP
8.6 Verify signatures of download instructions and DKM with PDA certificate
8.7 Verify signatures of install instructions and IKM with PIA certificate
8.8 Assemble all data into VUUP content
8.9 Hash VUUP content
8.3 Request PDA and PIA certificates
8.4 Send PDA and PIA certificates
CMS
8.10 Request signature on the hash of the VUUP content
8.12 Send signature
PSS
8.11 Generate signature of hash
8.13 Upload signed VUUP and VCM certificate
8.14 Send status on VUUP upload
Vehicle Cloud Service

Figure 9.23: Diagram of the sub-problem *Package the Instructions*. Steps marked with * can occur in parallel.

(8.1)	$VCM \rightarrow PDA$	$Request(DownloadInstructions)$ $\parallel Request(DKM)$ $\parallel Request(PDA_{Cert})$
	$VCM \rightarrow PIA$	$Request(InstallationInstructions)$ $\parallel Request(IKM)$ $\parallel Request(PIA_{Cert})$
(8.2)	$PDA \rightarrow VCM$	$[SymEnc(DownloadInstructions,$ $DKM_{Key})]_{PDA}$ $\parallel [DKM]_{PDA} \parallel PDA_{Cert}$
	$PIA \rightarrow VCM$	$[SymEnc(InstallationInstructions,$ $IKM_{Key})]_{PIA}$ $\parallel [IKM]_{PIA} \parallel PIA_{Cert}$
(8.3)	$VCM \rightarrow CMS$	$Request(PDA_{Cert})$ $\parallel Request(PIA_{Cert})$
(8.4)	$CMS \rightarrow VCM$	$PDA_{Cert} \parallel PIA_{Cert}$
(8.10)	$VCM \rightarrow PSS$	$Hash(VUUP_{Content})$
(8.12)	$PSS \rightarrow VCM$	$Sign(Hash(VUUP_{Content}),$ $VCM_{PrivateKey})$
(8.13)	$VCM \rightarrow VCS$	$VUUP$
(8.14)	$VCS \rightarrow VCM$	$Success(VUUP)$

Table 9.10: Communication scheme for the sub-problem *Package the Instructions*, where VCS is the Vehicle Cloud Service.

$$\begin{aligned} S =&\{Vehicle_{PrivateKey},\ Root_{PrivateKey},\ PDA_{PrivateKey},\ PIA_{PrivateKey},\\ &PSA_{PrivateKey},\ VCM_{PrivateKey},\ DKM_{Key},\ IKM_{Key}\}\\ \mathcal{D} =&\{(VUUP_{URL},\ VehicleCloudService),\ (VUUP,\ VCM),\end{aligned}$$

For any execution of this sub-problem, the entities PDA, PIA, VCM, CMS, and PSS are aware of which update round it belongs to since these entities have been in the same update round in sub-problems *Order Initiation* and *Create Download Instructions*. To derive the specific sub-problem requirements from the system requirements (see Section 9.3.5), we define S and $\mathcal{D}$ as seen above, and the partial order $\mathcal{P}(\ell) = \{\ell_1 < \ell_3,\ \ell_2 < \ell_3,\ \ell_3 < \ell_4,\ \ell_4 < \ell_5,\ \ell_5 < \ell_6,\ \ell_6 < \ell_7\}$. In other words, ℓ_1 and ℓ_2 happen parallel initially, followed sequentially by the remaining handling events, with ℓ_7 happening last.

ℓ_1: VCM receives Download Instructions and DKM from PDA.

ℓ_2: VCM receives Installation Instructions and IKM from PIA.

ℓ_3: VCM retrieves PDA and PIA certificates from CMS.

ℓ_4: VCM assembles the VUUP.

ℓ_5: PSS signs the VUUP.

ℓ_6: VCM uploads the signed VUUP along with the VCM certificate to Vehicle Cloud Service.

ℓ_7: Vehicle Cloud Service receives a status of the signed VUUP and VCM certificate being uploaded to Vehicle Cloud Service.

9.6.2.7 Step 9–11: Notify Order Ready

When the VUUP has been created, the Order Agent is notified that the order is ready for download (see step 9 in Figure 9.24). The notification consists of a signed URL to the VUUP file. The signature of this URL is validated by the Order Agent before it is uploaded to the Order Cloud Service (steps 10.1 and 10.2).

In step 11 of this sub-problem, CDA will pull the status from Order Cloud Service to see if any update is available. We assume updates are always available since if no updates are available, there will be an illegitimate termination of the update process.

The entities CDA, Order Agent, Order Cloud Service, and VCM have starting contexts because their listening tasks are included in a previous sub-problem (see sub-problem *Order Initiation*). VCM's starting context contains the $VUPP_{URL}$.

(9)	$VCM \rightarrow OrderAgent$	$[VUUP_{URL}]_{VCM}$
(10.2)	$OrderAgent \rightarrow OCS$	$[VUUP_{URL}]_{VCM}$
(10.3)	$OCS \rightarrow OrderAgent$	$Success(VUUP_{URL})$
(11.1)	$CDA \rightarrow OCS$	$Request(VUUP_{URL})$
(11.2)	$OCS \rightarrow CDA$	$[VUUP_{URL}]_{VCM} \parallel VCM_{Cert}$

Table 9.11: Communication scheme for the sub-problem *Notify Order Ready*, where OCS is the Order Cloud Service.

To derive the specific sub-problem requirements from the system requirements (see Section 9.3.5), we define $S = \{VCM_{PrivateKey}\}$, $\mathcal{D} =$

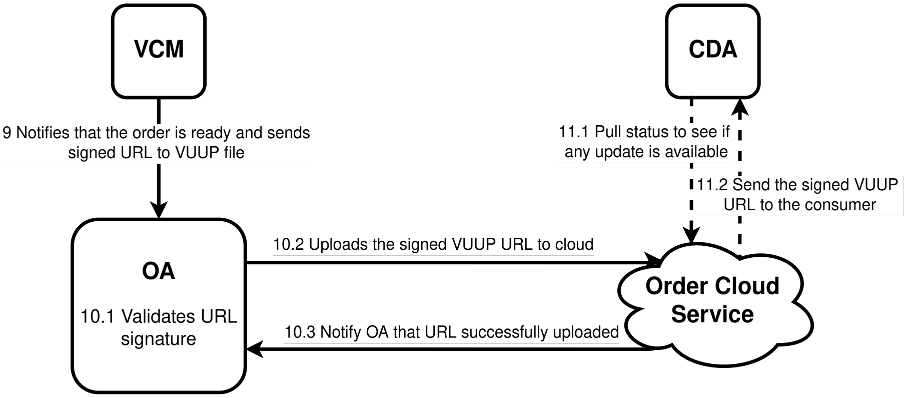

VCM
CDA
9 Notifies that the order is ready and sends signed URL to VUUP file
11.1 Pull status to see if any update is available
11.2 Send the signed VUUP URL to the consumer
OA
10.1 Validates URL signature
10.2 Uploads the signed VUUP URL to cloud
10.3 Notify OA that URL successfully uploaded
Order Cloud Service

Figure 9.24: Diagram of the sub-problem *Notify Order Ready.*

$\{(VUUP_{URL},\ VehicleCloudService),\ (VCM_{Cert},\ Root)\}$, and the partial order $\mathcal{P}(\ell) = \ell_i < \ell_{i+1}$ for $1 \leq i \leq 4$ with the following labels:

ℓ_1: VCM sends $[VUUP_{URL}]_{VCM}$ to Order Agent

ℓ_2: Order Agent sends $[VUUP_{URL}]_{VCM}$ to the Order Cloud Service

ℓ_3: CDA pulls status from the Order Cloud Service

ℓ_4: The Order Cloud Service sends $[VUUP_{URL}]_{VCM}$ to CDA

9.6.3 Decapsulation

We summarise the decapsulation stage defined by [31, Sec. 4.2] as follows: CDA retrieves a $VUUP_{URL}$ from Order Cloud Service, further used to retrieve the VUUP from Vehicle Cloud Service. Once CDA has validated the VUUP, CDA uses CSA for the decryption and plain text retrieval of the Download Instructions. The Download Instructions is used to download software from the Software Repository. After that, CIA will use CSA to decrypt and for the plain text retrieval of the Installation Instructions, and with the help of CSA, install software to the ECUs.

We have divided the decapsulation stage into five sub-problems. A summary of the division can be seen in Figure 9.25. In [31, Fig. 4], a more detailed version of the entire decapsulation process is presented.

9.6.3.1 Step 1–4: Download VUUP

Once the encapsulation is done (see Section 9.6.2), CDA retrieves the $VUUP_{URL}$, and uses the URL to obtain a valid VUUP (see step 2.2 in Figure 9.26 and Table 9.12). CDA also guarantees that the content of the VUUP is valid. For this sub-problem, CDA has a starting context because its initiation task is in a previous sub-problem (see Section 9.6.2.1). Order Cloud Service and the Vehicle Cloud Service also have starting contexts, which respectively contain the $VUUP_{URL}$ and VUUP for the update round (see Sections 9.6.2.6 and 9.6.2.7). Note that the first steps of this sub-problem, steps 1.1 and 1.2

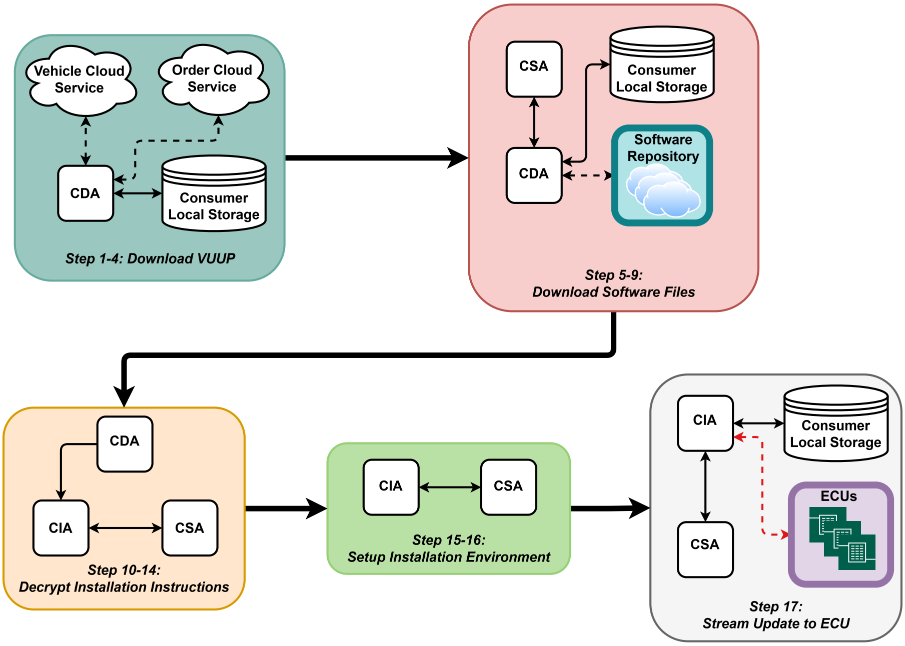


Figure 9.25: Using the specifications of the decapsulation stage provided by [31, Sec. 4.2], we present the following division of sub-problems.

in Figure 9.26 and Table 9.12, are the same as the last steps for the previous sub-problem (see Section 9.6.2.7).

(1.1)	$CDA \rightarrow OrderCloudService$	$Request(VUUP_{URL})$
(1.2)	$OrderCloudService \rightarrow CDA$	$[VUUP_{URL}]_{VCM} \parallel VCM_{Cert}$
(2.1)	$CDA \rightarrow VehicleCloudService$	$VUUP_{URL}$
(2.2)	$VehicleCloudService \rightarrow CDA$	$VUUP$
(2.3)	$CDA \rightarrow ConsumerLocalStorage$	$VUUP$
(2.4)	$ConsumerLocalStorage \rightarrow CDA$	$Success(VUUP)$

Table 9.12: Communication scheme for the sub-problem *Download Software Files.*

$$\begin{aligned}
S =&\{VUUP_{URL},\ DownloadInstructions,\ InstallationInstructions,\\
&DKM_{Key}, IKM_{Key},\ MKM_{Key},\ SKA_{Key},\ VCM_{PrivateKey},\\
&\ Vehicle_{PrivateKey},\ PDA_{PrivateKey}, PIA_{PrivateKey},\ PSA_{PrivateKey},\\
&Root_{PrivateKey}\}\\
\mathcal{D} =&\{(VUUP_{URL},\ VehicleCloudService), (VUUP,\ VCM),\\
&\ (DownloadInstructions,\ PDA), (InstallationInstructions,\ PIA),\\
&(DKM,\ PDA),\ (IKM,\ PIA),\ (VCM_{Cert},\ Root),\ (PDA_{Cert},\ Root),\\
&\ (PIA_{Cert},\ Root),\ (Root_{Cert},\ Root)\}
\end{aligned}$$

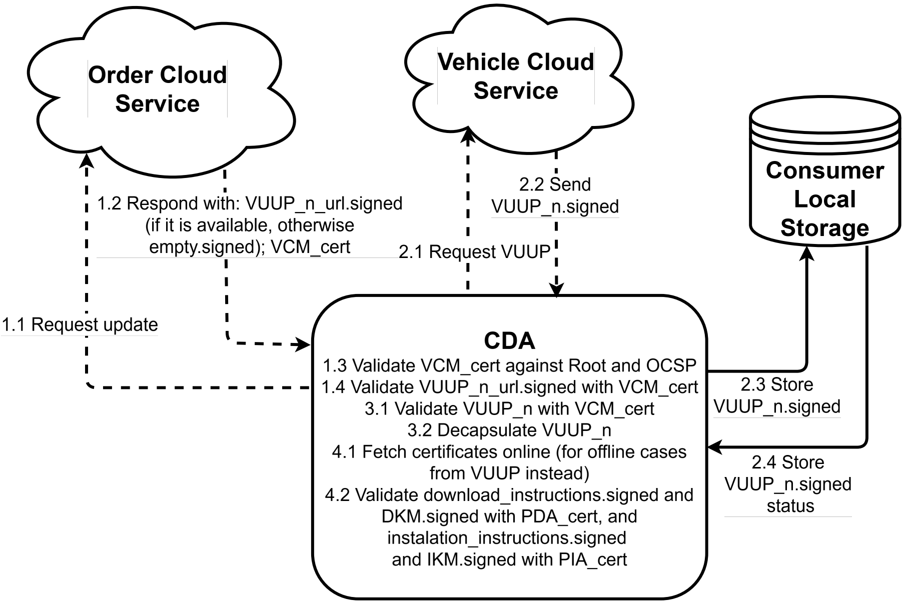

Order Cloud Service
Vehicle Cloud Service
Consumer Local Storage
1.2 Respond with: VUUP_n_url.signed (if it is available, otherwise empty.signed); VCM_cert
2.2 Send VUUP_n.signed
2.1 Request VUUP
1.1 Request update
CDA
1.3 Validate VCM_cert against Root and OCSP
1.4 Validate VUUP_n_url.signed with VCM_cert
3.1 Validate VUUP_n with VCM_cert
3.2 Decapsulate VUUP_n
4.1 Fetch certificates online (for offline cases from VUUP instead)
4.2 Validate download_instructions.signed and DKM.signed with PDA_cert, and instalation_instructions.signed and IKM.signed with PIA_cert
2.3 Store VUUP_n.signed
2.4 Store VUUP_n.signed status

Figure 9.26: Diagram of the sub-problem *Download VUUP*.

To derive the specific sub-problem requirements from the system requirements (see Section 9.3.5), we define S, $\mathcal{D}$ and the partial order $\mathcal{P}(\ell) = \ell_i < \ell_{i+1}$ for $1 \leq i \leq 4$ with the following labels:

ℓ_1: Order Cloud Service has sent the signed $VUUP_{URL}$.

ℓ_2: CDA has validated the $VUUP_{URL}$.

ℓ_3: Vehicle Cloud Service has sent the signed VUUP.

ℓ_4: Consumer Local Storage has stored the signed VUUP.

ℓ_5: CDA has validated Download Instructions, DKM, Installation Instructions and IKM.

9.6.3.2 Step 5–9: Download Software Files

CDA requests CSA to associate master keys in DKM to specific trusted applications within the Trusted Execution Environment (TEE). This is followed by the decryption of the Download Instructions on behalf of CDA. Then CDA uses the download instructions to retrieve the software from Software Repository (see step 9.1 in Figure 9.27 and Table 9.13). For this sub-problem, CDA and Consumer Local Storage have starting contexts because their respective initiation and listening tasks are included in previous sub-problems (see Sections 9.6.2.1 and 9.6.3.1). CDA's starting context contains the signed DKM, PDA_{Cert}, VCM_{Cert}, and the signed and encrypted Download Instructions (see Section 9.6.3.1).

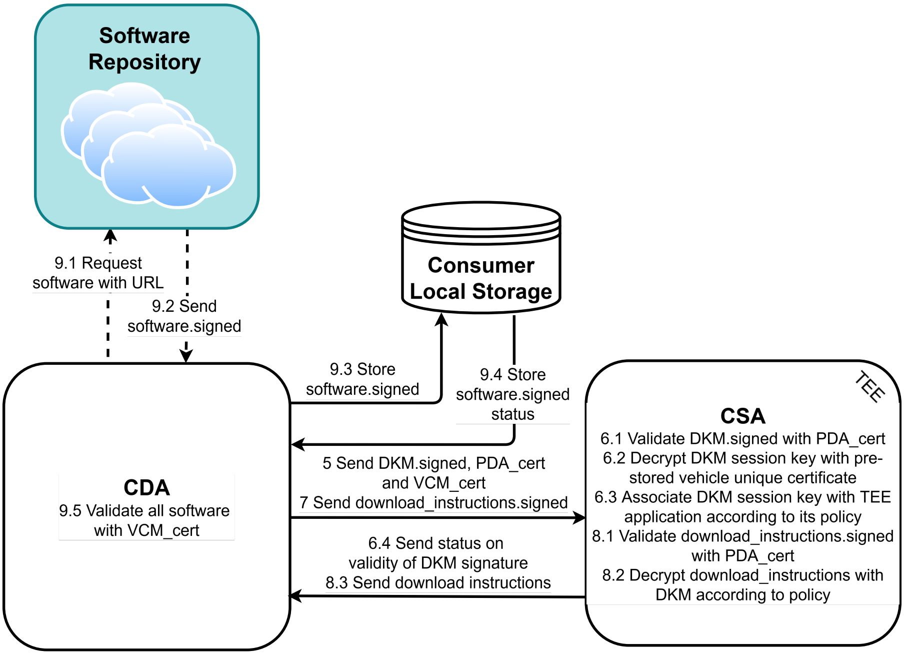

Software Repository
9.1 Request software with URL
9.2 Send software.signed
Consumer Local Storage
9.3 Store software.signed
9.4 Store software.signed status
CDA
9.5 Validate all software with VCM_cert
5 Send DKM.signed, PDA_cert and VCM_cert
7 Send download_instructions.signed
6.4 Send status on validity of DKM signature
8.3 Send download instructions
TEE
CSA
6.1 Validate DKM.signed with PDA_cert
6.2 Decrypt DKM session key with pre-stored vehicle unique certificate
6.3 Associate DKM session key with TEE application according to its policy
8.1 Validate download_instructions.signed with PDA_cert
8.2 Decrypt download_instructions with DKM according to policy

Figure 9.27: Diagram of the sub-problem *Download Software Files*.

$$
\begin{aligned}
S =&\{\mathit{Software},\ \mathit{DownloadInstructions},\ \mathit{DKM}_{\mathit{Key}},\ \mathit{SKA}_{\mathit{Software}_{\mathit{Key}}},\\
&\mathit{VCM}_{\mathit{PrivateKey}},\ \mathit{Vehicle}_{\mathit{PrivateKey}},\ \mathit{PDA}_{\mathit{PrivateKey}},\ \mathit{Supplier}_{\mathit{PrivateKey}},\\
&\mathit{Root}_{\mathit{PrivateKey}}\}\\
\mathcal{D} =&\{(\mathit{Software},\ \mathit{Supplier}),\ (\mathit{DownloadInstructions},\ \mathit{PDA}),\\
&(\mathit{DKM},\ \mathit{PDA}),\ (\mathit{VCM}_{\mathit{Cert}},\ \mathit{Root}),\ (\mathit{Vehicle}_{\mathit{Cert}},\ \mathit{Root}),\\
&(\mathit{PDA}_{\mathit{Cert}},\ \mathit{Root}),\ (\mathit{Root}_{\mathit{Cert}},\ \mathit{Root})\}
\end{aligned}
$$

To derive the specific sub-problem requirements from the system requirements (see Section 9.3.5), we define S, $\mathcal{D}$, and the partial order $\mathcal{P}(\ell) = \ell_i < \ell_{i+1}$ for $1 \leq i \leq 5$ with the following labels:

ℓ_1: CSA has associated the DKM.

ℓ_2: CSA decrypts the Download Instructions.

ℓ_3: CDA has received the decrypted Download Instructions.

ℓ_4: Software Repository has sent $\mathit{Software}_{\mathit{Encased}}$.

ℓ_5: Consumer Local Storage has stored $\mathit{Software}_{\mathit{Encased}}$.

ℓ_6: CDA has validated $\mathit{Software}_{\mathit{Encased}}$.

(5)	$CDA \rightarrow CSA$	$[DKM]_{PDA} \parallel PDA_{Cert}$ $\parallel VCM_{Cert}$
(6.4)	$CSA \rightarrow CDA$	$Success(DKM)$
(7)	$CDA \rightarrow CSA$	$[SymEnc($ $DownloadInstructions,$ $DKM_{Key})]_{PDA}$
(8.3)	$CSA \rightarrow CDA$	$DownloadInstructions$
(9.1)	$CDA \rightarrow SoftwareRepository$	$Software_{URL}$
(9.2)	$SoftwareRepository \rightarrow CDA$	$Software_{Encased}$
(9.3)	$CDA \rightarrow ConsumerLocalStorage$	$Software_{Encased}$
(9.4)	$ConsumerLocalStorage \rightarrow CDA$	$Success(Software)$

Table 9.13: Communication scheme for the sub-problem *Download Software Files*. Note that the VCM_{Cert} is sent to CSA in step 5, so that CSA has access to the certificate in step 17 (see Section 9.6.3.5) when it needs to validate software signatures [308].

9.6.3.3 Step 10–14: Decrypt Installation Instructions

On behalf of CIA, CSA initiates the IKM, followed by the decryption of Installation Instructions (see steps 12.5 and 14.3 in Figure 9.28 and Table 9.14). For this sub-problem, CDA and CSA have starting contexts because their respective initiation and listening tasks are run in previous sub-problems (see Sections 9.6.3.1 and 9.6.3.2). CDA's starting context contains PIA_{Cert}, the signed IKM, and the signed and encrypted Installation Instructions (see Section 9.6.3.1).

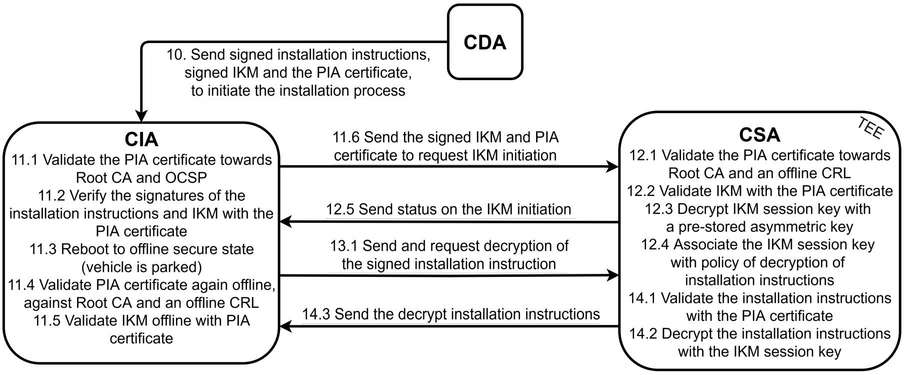

CDA
10. Send signed installation instructions, signed IKM and the PIA certificate, to initiate the installation process
CIA
11.1 Validate the PIA certificate towards Root CA and OCSP
11.2 Verify the signatures of the installation instructions and IKM with the PIA certificate
11.3 Reboot to offline secure state (vehicle is parked)
11.4 Validate PIA certificate again offline, against Root CA and an offline CRL
11.5 Validate IKM offline with PIA certificate
11.6 Send the signed IKM and PIA certificate to request IKM initiation
12.5 Send status on the IKM initiation
13.1 Send and request decryption of the signed installation instruction
14.3 Send the decrypt installation instructions
CSA
TEE
12.1 Validate the PIA certificate towards Root CA and an offline CRL
12.2 Validate IKM with the PIA certificate
12.3 Decrypt IKM session key with a pre-stored asymmetric key
12.4 Associate the IKM session key with policy of decryption of installation instructions
14.1 Validate the installation instructions with the PIA certificate
14.2 Decrypt the installation instructions with the IKM session key

Figure 9.28: Diagram of the sub-problem *Decrypt Installation Instructions*.

$$\begin{aligned}
S =& \{InstallationInstructions,\ IKM_{Key},\ MKM_{Key},\ SKA_{Key},\\
& PIA_{PrivateKey},\ PSA_{PrivateKey},\ Root_{PrivateKey}\}\\
\mathcal{D} =& \{(InstallationInstructions,\ PIA),\ (IKM,\ PIA),\ (PIA_{Cert},\ Root),\\
& (Root_{Cert},\ Root)\}
\end{aligned}$$

(10)	$CDA \rightarrow CIA$	$[SymEnc(InstallationInstructions, IKM_{Key})]_{PIA} \parallel [IKM]_{PIA} \parallel PIA_{Cert}$
(11.6)	$CIA \rightarrow CSA$	$[IKM]_{PIA} \parallel PIA_{Cert}$
(12.5)	$CSA \rightarrow CIA$	$Success(IKM)$
(13.1)	$CIA \rightarrow CSA$	$[SymEnc(InstallationInstructions, IKM_{Key})]_{PIA}$
(14.3)	$CSA \rightarrow CIA$	$InstallationInstructions$

Table 9.14: Communication scheme for the sub-problem *Setup Installation Environment.*

To derive the specific sub-problem requirements from the system requirements (see Section 9.3.5), we define S, $\mathcal{D}$ and the partial order $\mathcal{P}(\ell) = \ell_i < \ell_{i+1}$ for $1 \leq i \leq 3$ with the following labels:

ℓ_1: CIA initiates offline mode.

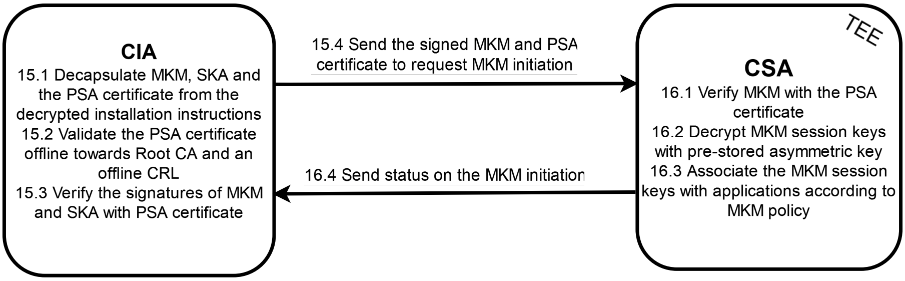
ℓ_2: CSA associates the IKM session key with its policy.

ℓ_3: CSA decrypts the Installation Instructions.

ℓ_4: CIA receives the decrypted Installation Instructions.

9.6.3.4 Step 15–16: Setup Installation Environment

On behalf of CIA (see Step 15.4 in Figure 9.29 and Table 9.15), CSA sets up an installation environment using the MKM. Afterwards, CSA is ready to decrypt software and unlock the ECUs (see Section 9.6.3.5). For this sub-problem, CSA and CIA have starting contexts because their listening tasks are run in previous sub-problems (see Sections 9.6.3.2 and 9.6.3.3 respectively). CIA's starting context has access to a set of decrypted Installation Instructions (see Section 9.6.3.3).



15.1 Decapsulate MKM, SKA and the PSA certificate from the decrypted installation instructions

15.2 Validate the PSA certificate offline towards Root CA and an offline CRL

15.3 Verify the signatures of MKM and SKA with PSA certificate

15.4 Send the signed MKM and PSA certificate to request MKM initiation

16.1 Verify MKM with the PSA certificate

16.2 Decrypt MKM session keys with pre-stored asymmetric key

16.3 Associate the MKM session keys with applications according to MKM policy

16.4 Send status on the MKM initiation

Figure 9.29: Diagram of the sub-problem *Setup Installation Environment.*

(15.4)	$CIA \rightarrow CSA$	$[MKM]_{PSA} \parallel PSA_{Cert}$
(16.4)	$CSA \rightarrow CIA$	$Success(MKM)$

Table 9.15: Communication scheme for the sub-problem *Setup Installation Environment.*

$$
\begin{aligned}
S =&\{InstallationInstructions,\ MKM_{Key},\ SKA_{Key},\ PSA_{PrivateKey},\\
&Root_{PrivateKey}\}\\
\mathcal{D} =&\{(InstallationInstructions,\ PIA),\ (SKA,\ PSA),\ (MKM, PSA),\\
&(PSA_{Cert}, Root),\ (Root_{Cert}, Root)\}
\end{aligned}
$$

To derive the specific sub-problem requirements from the system requirements (see Section 9.3.5), we define S, $\mathcal{D}$ and the partial order $\mathcal{P}(\ell) = \ell_i < \ell_{i+1}$ for $1 \leq i \leq 2$ with the following labels:

ℓ_1: CIA validates MKM and SKA.

ℓ_2: CSA associated the MKM keys with their policy.

ℓ_3: CIA receives MKM status from CSA.

9.6.3.5 Step 17: Stream Update to ECU

The goal of this sub-problem is to stream software updates to ECUs. This can mainly be divided into two processes; first, an ECU needs to be unlocked via a challenge-response schema, i.e., security access. The CSA solves the unlocking on behalf of CIA (see step 17.1 – 17.9 in Figure 9.30 and Table 9.16). Second, the software is transmitted to the ECU from CIA, after it has been decrypted by the CSA. The software signature is validated, and depending on whether the software was successfully installed or not, a status message is sent to CIA (see steps 17.12 – 17.21). These two steps are then repeated to install different software on different ECUs [31, Fig. 4]. However, as mentioned in Section 9.4.2.5 we only consider the update of one single ECU for one occasion.

For this sub-problem, Consumer Local Storage, CSA, and CIA have starting contexts because their respective initiation and listening tasks are run in previous sub-problems (see Sections 9.6.3.1 to 9.6.3.3). Consumer Local Storage's starting context contains $Software_{Encased}$. CSA's starting context contains $MKM_{SecurityAccess_{Key}}$, $MKM_{Software_{Key}}$ and VCM_{Cert}. CIA's starting context contains $SKA_{SecurityAccess_{Key}}$, $SKA_{Software_{Key}}$, $MKM_{SecurityAccess_{Key}}$ and $MKM_{Software_{Key}}$.

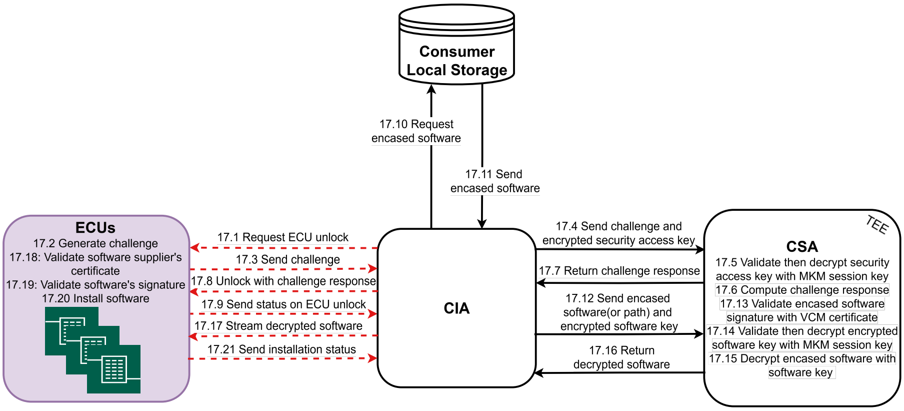

Consumer Local Storage
17.10 Request encased software
17.11 Send encased software
ECUs
17.2 Generate challenge
17.18: Validate software supplier's certificate
17.19: Validate software's signature
17.20 Install software
17.1 Request ECU unlock
17.3 Send challenge
17.8 Unlock with challenge response
17.9 Send status on ECU unlock
17.17 Stream decrypted software
17.21 Send installation status
CIA
17.4 Send challenge and encrypted security access key
17.7 Return challenge response
17.12 Send encased software(or path) and encrypted software key
17.16 Return decrypted software
TEE
CSA
17.5 Validate then decrypt security access key with MKM session key
17.6 Compute challenge response
17.13 Validate encased software signature with VCM certificate
17.14 Validate then decrypt encrypted software key with MKM session key
17.15 Decrypt encased software with software key

Figure 9.30: Diagram of the sub-problem *Stream Update to ECU*.

$$
\begin{aligned}
S =&\{SKA_{Software_{Key}},\ SKA_{SecurityAccess_{Key}},\ MKM_{Software_{Key}},\\
&MKM_{SecurityAccess_{Key}},\ Vehicle_{PrivateKey},\ Supplier_{PrivateKey},\\
&VCM_{PrivateKey},\ Root_{PrivateKey}\}\\
\mathcal{D} =&\{(Software,\ Supplier),\ (SKA_{Software_{Key}},\ PSA),\\
&(SKA_{SecurityAccess_{Key}},\ PSA),\ (MKM_{Software_{Key}},\ PSA),\\
&(MKM_{SecurityAccess_{Key}},\ PSA),\ (Supplier_{Cert},\ Root),\ (VCM_{Cert},\ Root),\\
&(Root_{Cert},\ Root)\}
\end{aligned}
$$

To derive the specific sub-problem requirements from the system requirements (see Section 9.3.5), we define S, $\mathcal{D}$ and the partial order $\mathcal{P}(\ell) = \{\ell_1 < \ell_2, < \ell_3 < \ell_4 < \ell_9,\ \ell_5 < \ell_6, < \ell_7 < \ell_8 < \ell_9\}$. In other words, both sequences ℓ_{1-4} and ℓ_{5-8} happen before ℓ_9 but can be executed concurrently.

ℓ_1: ECU generates challenge.

ℓ_2: CIA forwards the challenge to CSA.

ℓ_3: CSA responds to the challenge.

ℓ_4: ECU accepts the challenge.

ℓ_5: Consumer Local Storage sends $Software_{Encased}$ to CIA.

ℓ_6: CIA receives the $Software_{Encased}$ from Consumer Local Storage.

ℓ_7: CIA sends the $Software_{Encased}$ and $Software_{Key}$ to CSA.

ℓ_8: CSA decrypts software.

ℓ_9: ECU installs software.

(17.1)	$CIA \rightarrow ECU$	$Request(ECU)$
(17.3)	$ECU \rightarrow CIA$	$ECU_{Challenge}$
(17.4)	$CIA \rightarrow CSA$	$ECU_{Challenge}$ $\parallel AuthSymEnc($ $SKA_{SecurityAccess_{Key}},$ $MKM_{SecurityAccess_{Key}})$
(17.7)	$CSA \rightarrow CIA$	$ECU_{Challenge-Response}$
(17.8)	$CIA \rightarrow ECU$	$ECU_{Challenge-Response}$
(17.9)	$ECU \rightarrow CIA$	$ECU_{Unlocked}$
(17.10)	$CIA \rightarrow ConsumerLocalStorage$	$Request(Software)$
(17.11)	$ConsumerLocalStorage \rightarrow CIA$	$Software_{Encased}$
(17.12)	$CIA \rightarrow CSA$	$Software_{Encased}$ $\parallel AuthSymEnc($ $SKA_{Software_{Key}},$ $MKM_{Software_{Key}})$
(17.16)	$CSA \rightarrow CIA$	$[Software]_{Supplier}$
(17.17)	$CIA \rightarrow ECU$	$[Software]_{Supplier}$
(17.21)	$ECU \rightarrow CIA$	$ECU_{Installation-Status}$

Table 9.16: Communication scheme for the sub-problem *Stream Update to ECU*. Note that this sub-problem can be repeated [31, Fig.4], and therefore $SKA_{SecurityAccess_{Key}}$, $SKA_{Software_{Key}}$ and $Software$ might refer to different keys and software for different iterations.

9.7 Methods

We present the methods for simulating our assumptions and system settings (see Section 9.3) in ProVerif. Furthermore, we outline the methods used to model our requirements. We also formally demonstrate our proofs for intra-round uniqueness and termination, i.e., System-level Requirements 9.3.5.4 and 9.3.5.6.

9.7.1 Simulation of Cryptographic Primitives in ProVerif

In this section, the simulation of cryptographic primitives is described.

9.7.1.1 Unauthenticated Symmetric Encryption

Blanchet et al. [288] illustrates an example of unauthenticated symmetric encryption, as seen in Listing 9.1, by defining functions for symmetric encryption `senc` and symmetric decryption `sdec`. Both functions take arguments of a `bitstring` (a built-in type) and a `key`. Note that `key` is a user-defined type that represents symmetric keys. The functions `senc` and `sdec` output the ciphertext and plaintext, respectively. The equations on lines 4 and 5 describe the relationship between the functions `senc` and `sdec`, where `m` is the message and `k` is the key. These equations ensure that whenever the algorithm decrypts some data, `sdec` outputs the original plaintext if and only if the same key was used during encryption and the ciphertext has not been modified. Otherwise, it outputs some arbitrary data.

```
type key.
fun senc(bitstring, key): bitstring.
fun sdec(bitstring, key): bitstring.
equation forall m: bitstring, k :key; sdec(senc(m,k), k) = m.
equation forall m: bitstring, k :key; senc(sdec(m,k), k) = m.
```

Listing 9.1: Unauthenticated symmetric encryption [288, Sec. 4.2.2].

9.7.1.2 Authenticated Symmetric Encryption

The authenticated encryption in Listing 9.2 ensures that the same key is used for encryption and decryption and that modified ciphertexts are detected. In ProVerif, we model this by defining the decryption as a destructor through the reserved word `reduc` [288, Sec. 3.1]. If the mentioned abnormalities are detected during the decryption, ProVerif blocks. Alternatively, the `authSdec`: `let m = authSdec(c) in P else Q` syntax can be used, such that process `P` is run if decryption succeeded and `Q` is run on failure. If `else Q` is omitted, then the process terminates.

```
fun authSenc(bitstring, key): bitstring.
reduc forall m: bitstring, k: key; authSdec(authSenc(m, k), k) = m.
```

Listing 9.2: Authenticated symmetric encryption [288, Sec. 3.1.2].

9.7.1.3 Asymmetric Encryption

Asymmetric encryption (see Listing 9.3) is similar to authenticated symmetric encryption. The main difference is in the consideration of key pairs. In ProVerif, the keypair is modeled such that the public key can be retrieved from the private key, but not the other way around [288]. Also, a message decrypted with a private key must have been encrypted with the corresponding public key.

```
type skey.
type pkey.
fun pk(skey): pkey.
fun aenc(bitstring, pkey): bitstring.
reduc forall m: bitstring, k: skey; adec(aenc(m, pk(k)), k) = m.
```

Listing 9.3: Asymmetric encryption [288, Sec. 3.1.2].

9.7.1.4 Hash Function

The hash function (see Listing 9.4) takes a `bitstring` as input and outputs an arbitrary `bitstring`, and has no associated destructors or equations [288, Sec. 4.2.5]. By excluding destructors and equations, the hash function resembles a random oracle model, making it impossible to reverse the hash to obtain the original value.

```
fun hash(bitstring): bitstring.
```

Listing 9.4: Hash function [288, Sec. 4.2.5].

9.7.1.5 Digital Signature

The modeling of digital signatures can be more complex in comparison to other cryptographic primitives. Blanchet et al. [288] describe a method to simulate the signing function. It is modeled as a function that takes a message of type `bitstring` and a signing private key of type `sskey` as input and outputs the signed message as a `bitstring` (see Listing 9.5). A reducer is used to validate the authenticity of a signed message. The reducer is specified to pattern match on the correct private key, similar to how asymmetric encryption is simulated.

```
type sskey.
type spkey.

fun spk(sskey): spkey.
fun sign(bitstring, sskey): bitstring.

reduc forall m: bitstring, k: sskey; getmess(sign(m, k)) = m.
reduc forall m: bitstring, k: sskey; checksign(sign(m, k), k) = m.
```

Listing 9.5: Digital signatures schema presented in the ProVerif manual [288, Sec. 3.1.2.].

However, UniSUF requires a more complex signing schema, since data signing in UniSUF considers multiple steps and messages. Therefore, the previously stated method of signing function fails to capture all our use cases.

We specify a more complex signing schema (see Listing 9.6). This schema allows for building a signature step-by-step:

- The `sgnHash` is used to sign a hash.
- The `createSgn` function takes the signed hash of a message and appends it to the message.
- The `validateSgn` function checks that a properly signed message contains the message and the signed hash of the message.
- The `equation` allows the more traditional `sgn` signing function to be used interchangeably with `createSgn`.

In summary, `sgnHash` and `createSgn` are used to properly model the signing process in UniSUF, while `sgn` serves as a shorthand to create signatures.

9.7.1.6 Certificates

For certificates, we use a simplified version of the implementation presented by [297, Appendix A.1]. On line 2 (see Listing 9.7), the `createCert` function outputs a certificate from a signing public key and a signing private key. We

```
type sskey.
type spkey.
fun spk(sskey): spkey.

fun sgnHash(bitstring, sskey): bitstring.
fun sgn(bitstring, sskey): bitstring.
fun createSgn(bitstring, bitstring): bitstring.

equation forall m: bitstring, ssk: sskey;
    createSgn(m, sgnHash(hash(m), ssk)) = sgn(m, ssk).
reduc forall m: bitstring, k: sskey;
    validateSgn(createSgn(m, sgnHash(hash(m), k)), spk(k)) = m.
```

Listing 9.6: Digital signatures [288, Sec. 3.1.2].

define two destructors `validateCert` and `getCert` for the function `createCert`. The destructor `validateCert` validates whether the public key corresponds to the entity that issued the certificate, and outputs the holder's public key if and only if this validation passes. The destructor `getCert` is similar to `validateCert`, except that `getCert` does not validate the certificate.

```
type cert.
fun createCert(spkey, sskey): cert.
reduc forall holderSpk: spkey, issuerSsk: sskey;
    validateCert(createCert(holderSpk, issuerSsk), spk(issuerSsk)) =
    ↪  (holderSpk, spk(issuerSsk)).
reduc forall holderSpk: spkey, issuerSsk: sskey;
    getCert(createCert(holderSpk, issuerSsk)) = (holderSpk, spk(issuerSsk)).
```

Listing 9.7: Certificates [297, Appendix A.1].

9.7.2 Representing Requirements in ProVerif

ProVerif allows assertions of the system properties that the model must fulfill. All such assertions are declared using the `query` keyword [288]. In this section, we explain our methods for specifying the security requirements of UniSUF in ProVerif.

9.7.2.1 Modeling Confidential Secrets

One of the main features of ProVerif is the ability to verify the secrecy of the variables [288]. For secrecy queries, such as `query attacker(new x).` , ProVerif will attempt to prove that there exists no reachable state in which the attacker can access the local variable `x`. This query is a shorthand for `not attacker(new x).` , which means the query will output true if there is no state where the attacker can obtain the secret. More complex queries can also be created. We use the signature function in Section 9.7.1.5 as an example: `query attacker(sgn(new x, new ssk)).` . This checks that `x` signed with the private key `ssk` cannot be learned by the adversary.

9.7.2.2 Modeling Integrity of Handling Events

ProVerif allows users to define events using the keyword `event` [288, Sec. 3.2.2]. These user events represent our handling events, with the event's name serving as the label (ℓ). Also, data can be associated with user events, as shown in lines 1–3 in Listing 9.8, thereby representing the cryptographic materials (d) in the handling events. The update round identifier (r) is also passed in as data to the user events. By using the same r for all events, we ensure that the execution stays the same during the update round. However, d is specified for each handling event in the execution.

```
1  event A(bitstring, bitstring).
2  event B(bitstring, bitstring).
3  event C(bitstring, bitstring).
4
5  query r: bitstring, d1: bitstring; d2: bitstring, d3:bitstring;
6      event(C(r, d1));
7      event(C(r, d1)) ==> event((B(r, d2)) ==> event(A(r, d3))).
```

Listing 9.8: Modeling integrity of handling events in ProVerif. Line 6 asserts that event `C` is reachable. The nested correspondence assertion in line 7 specifies that if the event `C` has happened, then event `B` must have occurred at an earlier time, and event `A` must have happened before event `B`.

The integrity of handling events requires that handling events are executed according to a specified partial order. ProVerif correspondence assestions [288, Sec. 3.2.2] represents this requirement because they specify relationships between the events ensuring they occur in the desired order. For example, the assertion `event(e1) ==> event(e2)` states that whenever event `e1` has occurred, `e2` must have occurred previously.

Correspondence assertions can be extended to model a chain of events by using nested correspondence assertions [288, Sec. 4.3.1]. The nested correspondence assertion in Listing 9.8 specifies that if event `C` has occurred, then event `B` must have occurred previously, and event `A` must have occurred before `B`. Note that line 6 is a reachability query that checks if the event `C` is reachable. If event `C` is never reached, then the entire nested correspondence assertion is vacuously true and, therefore, not meaningful. That is, any system will meet the requirement as long as it never executes `C`.

The partial order of handling events can be modeled by creating different queries for chains of correspondence assertions. Lines 1 and 2 in Listing 9.9 give an example of two independent sequences that can occur concurrently: `C` must have occurred before `B` and `E` must have occurred before `D`.

Line 3 in Listing 9.9 illustrates a method for asserting that multiple events have occurred before a specific event [288, Sec. 4.3.1]. By linking the events using conjunctions, the listed events must have happened before the target event but in no specific order. This enables modeling concurrent events in a sequence of events.

```
query event(B) ==> event(C).
query event(D) ==> event(E).
query event(A) ==> (event(B) && event(D)).
```

Listing 9.9: Code example of partially ordered events being modelled in ProVerif [288, Sec. 4.3.1]. Note that declarations of events are excluded to avoid clutter.

9.7.2.3 Modeling Integrity of Cryptographic Materials

The integrity of cryptographic materials requires that the materials being processed remain the same during a sub-problem (see Section 9.3.5). This can be modelled by extending the approach used for the integrity of handling events (see Section 9.7.2.2). By using correspondence assertions, we model the relationships between cryptographic materials that are consumed by different handling events. Because we look at relationships, we can verify that the materials remain unchanged even as they are involved in various cryptographic transformations.

The code snippet in Listing 9.10 demonstrates this approach. Line 3 specifies that the cryptographic materials `cm` have not been modified between the events. Additionally, by looking at the signing function `sgn` (see Section 9.7.1.5) and the private key `ssk`, we can see that the `cm` was signed by a specific private key. Namely, the key belonging to the public key we specify in `A` and `B`: `pk(ssk)`.

```
1  query cm: bitstring, ssk: sskey;
2      event(C(sgn(aenc(cm, spk(ssk)), ssk)));
3      event(C(sgn(aenc(cm, spk(ssk)), ssk))) ==> (event(B(aenc(cm, spk(ssk))))
       ↪  ==> (event(A(cm, pk(ssk))))).
```

Listing 9.10: Modeling integrity of cryptographic materials. The assertion verifies that if event `C` occurs (`cm` is encrypted and then signed), then event `B` must have occurred earlier (`cm` was encrypted), and event `A` must have happened even earlier (the original unencrypted `cm` was available).

Because we divide UniSUF into sub-problems (see Section 9.6), there are assertions where we need cryptographic materials that are not used in the sub-problem. For example, some data could have been previously signed in another sub-problem. In that case, we need to create the signature with a private key only available during the system setup (see Section 9.7.3.1), but not for any entities in the sub-problem. We create a special `event started(r, d).` that is executed during the system setup and contains cryptographic materials (d) not used in the sub-problem. This event allows us to make assertions with these materials.

As explained in Section 9.7.2.2 and in this section, both integrity requirements can be specified using nested correspondence assertions. Therefore, we use a single nested correspondence assertion to specify the requirements for both handling events and the cryptographic materials.

9.7.2.4 Modeling Inter-Round Uniqueness

Informally, inter-round uniqueness means that there should not be two distinct update rounds that produce the same data. We can utilize ProVerif's correspondence assertions to verify this property, similar to the approach used by Wang [297]. The code snippet in Listing 9.11 demonstrates how the inter-round uniqueness property can be expressed in ProVerif. In this example, line 2 specifies a pre-condition that checks if it can reach both instances of the event `A`. Since both instances of `A` produce the same data, it should not be possible for them to originate from two distinct update rounds. Therefore, the assertion on line 3 requires that if both events are reachable, they must have occurred within the same update round.

```
1  query round1: bitstring, round2: bitstring, data: bitstring;
2      event(A(round1, data)) && event(A(round2, data))
3      ==> round1 = round2.
```

Listing 9.11: Code example of the inter-round uniqueness query for event `A`.

9.7.3 Simulation of System Settings and Assumptions

We describe the techniques for simulating the assumptions listed in Section 9.3.

9.7.3.1 Setting Up Cryptographic Materials and Starting Contexts

As mentioned in Section 9.4.2, we consider a system with a single producer responsible for producing updates for multiple vehicles. Additionally, vehicles occasionally need to update their software with the latest versions, i.e., each vehicle can be updated multiple times. While some of the cryptographic materials we identify in Section 9.4.1 are used for multiple updates, others are ephemeral, i.e., they can only be used during their designated update round. Next, we show how to simulate the relationships between update rounds and cryptographic materials.

We present the Mapping Tree in Figure 9.31, which is a tree consisting of different processes. The root *process* invokes multiple vehicles in its child processes *setupVehicle*, and each vehicle has multiple update rounds (invoked in the child process *setupUpdateRound*). In each process, we create cryptographic materials. Because the materials for each sub-problem vary, we create a tree for each sub-problem. Additionally, the sub-problems related to software preparation (see Section 9.6.1) do not consider vehicles. Therefore, *setupVehicle* is not included for these sub-problems. Instead, *process* directly invokes *setupUpdateRound*. Moreover, for *Secure Software Files* (see Section 9.6.1.1) and *Order Initiation* (see Section 9.6.2.1), *setupUpdateRound* is omitted. This is because these sub-problems contain the initiation task, meaning the update round of the current execution has not been initiated (see Section 9.5).

In the root *process*, we create the software supplier certificate and all producer certificates because these are used in all update rounds. However, the vehicle certificate is only used in update rounds for its vehicle. Therefore, this certificate is created in *setupVehicle*. All other cryptographic materials are

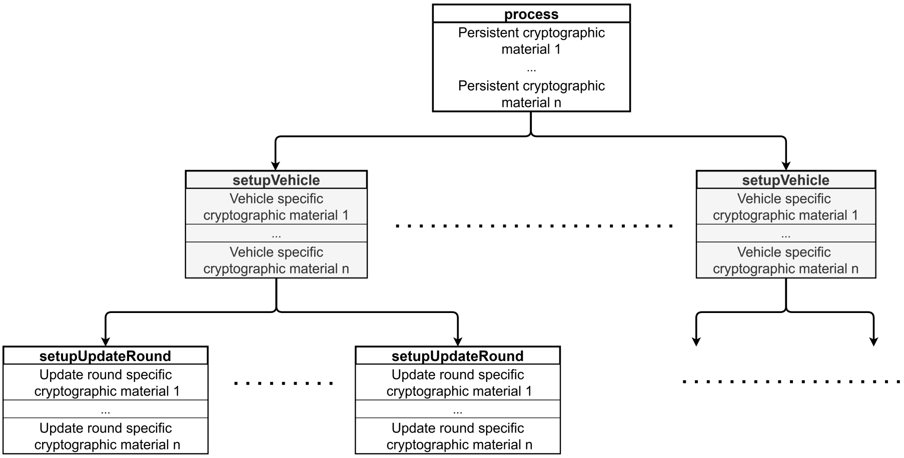

process
Persistent cryptographic material 1
...
Persistent cryptographic material n
setupVehicle
Vehicle specific cryptographic material 1
...
Vehicle specific cryptographic material n
setupVehicle
Vehicle specific cryptographic material 1
...
Vehicle specific cryptographic material n
setupUpdateRound
Update round specific cryptographic material 1
...
Update round specific cryptographic material n
setupUpdateRound
Update round specific cryptographic material 1
...
Update round specific cryptographic material n

Figure 9.31: Mapping Tree creates multiple update rounds for each vehicle. The tree also ensures that cryptographic materials are used in their assigned update round.

created in *setupUpdateRound* because they are only used in a single update round.

Each process passes down its cryptographic materials to its children processes. Therefore, *setupUpdateRound* can access cryptographic materials from *setupVehicle* and *process*, while *setupVehicle* can access materials from the root *process*.

9.7.3.2 Mapping Starting Contexts to Cryptographic Materials

As mentioned in Section 9.5, an entity participating in a sub-problem can have a starting context. This occurs when the entity has previously run either the initiation or listening task for another sub-problem in the same update round. We, therefore, extend our mapping tree from Section 9.7.3.1 to account for starting contexts (see Figure 9.32).

All producer and software repository entities with no starting context are invoked by *process*. Therefore, they only have access to the cryptographic materials used in all update rounds, i.e., they cannot access materials coupled to a vehicle or update round. Consumer entities with no starting context also have access to vehicle-specific materials and are therefore invoked by *setupVehicle*.

An entity with a starting context is invoked by *setupUpdateRound*. These entities can access the cryptographic materials created for the update round. We make sure to specify the materials in accordance with Section 9.6, such that only materials previously created in previous sub-problems are available.

Note that we create v_{id} and t_e (see Section 9.3.4) in *setupVehicle* and *setupUpdateRound* respectively. This allows us to separate the different update rounds from each other.

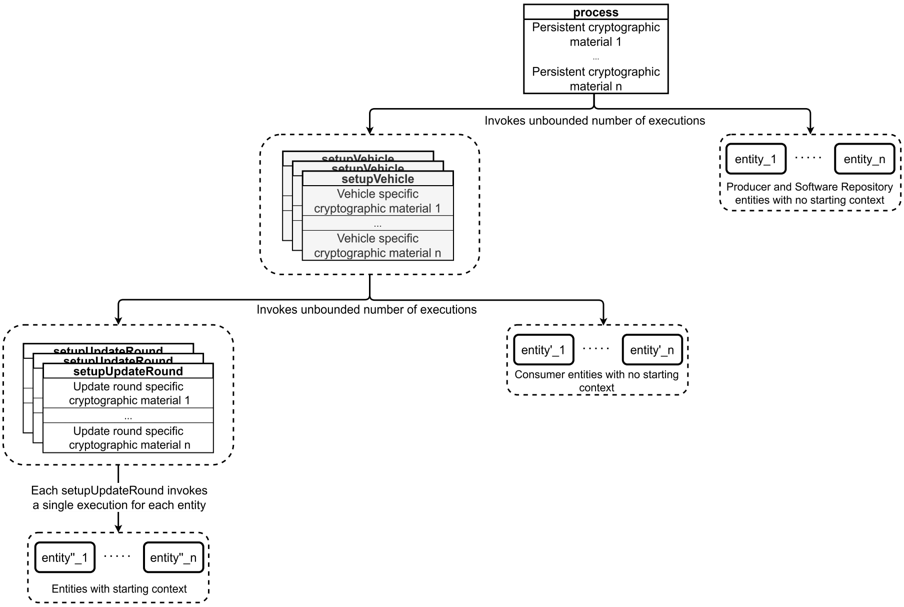

process
Persistent cryptographic material 1
...
Persistent cryptographic material n
Invokes unbounded number of executions
setupVehicle
setupVehicle
setupVehicle
Vehicle specific cryptographic material 1
...
Vehicle specific cryptographic material n
entity_1
entity_n
Producer and Software Repository entities with no starting context
Invokes unbounded number of executions
setupUpdateRound
setupUpdateRound
setupUpdateRound
Update round specific cryptographic material 1
...
Update round specific cryptographic material n
entity'_1
entity'_n
Consumer entities with no starting context
Each setupUpdateRound invokes a single execution for each entity
entity''_1
entity''_n
Entities with starting context

Figure 9.32: Extension of the mapping tree in Figure 9.31. This tree invokes the starting contexts for entities in the tree's sub-problem. The entities with starting contexts receive them when they are invoked by the tree contexts.

9.7.3.3 ProVerif Simulation of the Mapping Tree with Update Rounds

ProVerif uses a single main process, defined with the reserved word `process`, and sub-processes can be defined as macros by using the reserved word `let` [288, Sec. 3.1]. We use these two reserved words to implement our mapping tree (see Figure 9.32), as seen in Listing 9.12. For easier comprehension, the names of the sub-processes follow the mapping tree's structure.

Note the exclamation operator, `!`, in the listing, which in ProVerif invokes an unbounded number of replications of a process [288, Sec. 3.1.4]. This operator corresponds to the arrows we mark with *Invokes unbounded amount of processes* in our tree. By using an unbounded amount of invocation per process, we ensure that each vehicle receives multiple updates.

To give an execution of an entity access to the cryptographic materials discussed in Section 9.7.3.2, we pass the materials as parameters when invoking the execution. In ProVerif, this is achieved by: `entity_1(cryptographicMaterial)` [288, Sec. 3.1].

Since all public cryptographic materials are available to all, the adversary is explicitly made aware of them. To simulate this, we send the public material out on a channel that the adversary can read on, as such [288, Sec. 3.1]:

```
    out(publicChannel, (publicData1, publicData2, ..., publicDataN))
```

Note that this is done for all public cryptographic materials created in *process*, *setupVehicle* and *setupUpdateRound*.

```
let entity_1 ((*Parameters for entity_1*)) =
    (*Algorithm for entity_1....*).
(*
Define entity_n, entity'_1, entity'_n, entity''_1 and entity''_n as entity_1
↪  was defined
but with different parameters and algorithms.
*)

let setupVehicle((*Parameters for the vehicle*)) =
    (*Initiate Vehicle specific material*)
    !setupUpdateRound((*Send material to each update round*)) |
    !entity'_1((*Send initial material to entity'_1*)) |
    (* .... | *)
    !entity'_n((*Send initial material to entity'_n*)).

let setupUpdateRound((*Parameters needed for the update round*)) =
    (*Initiate update round specific material*)
    (*Running all entities in an update round*)
    entity''_1((*Send initial material to entity''_1*)) |
    (* .... | *)
    entity''_n((*Send initial material to entity''_n*)).

process
    (*Initiate persistent cryptographic material*)
    !setupVehicle((*Send initial material to each vehicle*)) |
    !entity_1((*Send initial material to entity_1*)) |
    (* .... | *)
    !entity_n((*Send initial material to entity_n*))
```

Listing 9.12: Code example of how entities are instantiated with their corresponding data, thereby simulating the update round mapping tree in Figure 9.32.

9.7.3.4 Reliable Communication

We simulate reliable communication in which the receiving-side messages are delivered according to the order in which the sender fetched them. The session identifier and message sequence number are included for all messages. The sequence number is incremented for each new message. This simulates a connection establishment and ensures the correct ordering of messages.

Our simulation is illustrated in Listing 9.13. Alice initiates the session by sending a message that includes a session identifier, data, and sequence number 1 (line 8). The session identifier tracks the session, while the hardcoded sequence number orders messages within it.

In line 14, Bob listens to the first message of any session by using the ProVerif pattern matching operator (=) [288, Sec. 3.1.2]. When Bob receives such a message, we check if another execution of Bob is already in the session. This is because we use an unbounded number of processes (see Section 9.7.3.3), which means that multiple executions of Bob can join the session. To prevent another execution of the same process from joining the same session, we adapt Wang's solution [297, Sec. 9.3.2].

```
1   free alice_bob: channel.
2
3   table bobSessions(bitstring, bitstring).
4
5   let Alice() =
6       new sessionId: bitstring; (* initiate the session identifier *)
7       new data1: bitstring; (* data to be sent *)
8       out(alice_bob, (sessionId, data1, 1));
9       in(alice_bob, (=sessionId, data2: bitstring, =2)).
10
11  let Bob() =
12      new processId: bitstring; (* each instance of Bob has a unique process ID
        ↪  *)
13      (* receive the session ID and data from Alice *)
14      in(aliceBob, (sessionId: bitstring, data1: bitstring, =1));
15
16      insert bobSessions(sessionId, processId);
17      get bobSessions(=sessionId, processId': bitstring) suchthat processId <>
        ↪  processId' in
18      (* another execution of Bob is already in the session, so we abort *)
19          0
20      else (
21      (* no other execution of Bob is currently in the session, so we proceed *)
22          new data2: bitstring;
23          out(alice_bob, (sessionId, data2, 2))
24      ).
```

Listing 9.13: Simulation of reliable communication in ProVerif.

Specifically, in line 3, we set up a table to store the session and the process identifiers. Then, in lines 16-17, we add the session identifier and Bob's process identifier to the table. We then check if another instance of Bob is already in the session, i.e., if there are at least two different process identifiers for the given session. If so, we abort the session establishment process. Otherwise, Bob proceeds to join the session. Wang [297, Sec. 9.3.2] states that this solution works because ProVerif schedules the processes such that, at most, one execution is allowed to proceed for a given session. Furthermore, ProVerif explores all possible schedules, including the schedules in which only a single execution is permitted to continue.

9.7.3.5 Secure and Reliable Communication

We enhance the reliable communication channel with security guarantees. In ProVerif, based on the Dolev-Yao threat model, the adversary has full control over the communication channels [288, Ch. 3]. They can freely read, update, or insert channel messages. This means messages are vulnerable to eavesdropping and tampering by an attacker. To prevent the adversary from reading, updating, or inserting channel messages, we declare the channel as private [288, Sec. 6.7.4]: `free c: channel[private].` .

9.7.3.6 Update Rounds

We describe how we simulate update rounds.

9.7.3.7 Replacing Session Identifiers With Update Round Identifiers

We include the update round identifier in all messages, as mentioned in Section 9.3.4. The update round identifier provides context to the cryptographic materials transmitted in our code and removes the need for the session identifiers discussed in Section 9.7.3.4. This is because an update round encapsulates multiple peer-to-peer sessions. This change is simple to implement. Instead of using session identifiers (`sessionId`), we use the update round identifier (see Section 9.3.4), i.e., VIN and the expiration time (`vin, expirationTime`), or just the expiration time (`expirationTime`).

9.7.3.8 Simulating the Listening Task in ProVerif

As mentioned in Section 9.5, the first task of all listeners is the listening task. We simulate this task so that listeners can discover new update rounds. As mentioned in Section 9.7.3.4, we adapt a solution by Wang [297, Sec. 9.3.2], to hinder multiple executions from working on the same session. For update rounds, we do the same, but we use a table containing update round identifiers instead of session identifiers.

Listeners that are considered producer and software repository entities are made aware of the round's VIN and expiration time, as follows:

```
in(c, (vin: bitstring, expirationTime: bitstring, ..., =1);
```

In contrast, consumer listeners are already aware of the VIN of their vehicle (see Section 9.3.1). Therefore, consumer listeners only learn the expiration time:

```
in(c, (=vin, expirationTime: bitstring, ..., =1));
```

Moreover, the equals operator (`=`), which is used for pattern matching in ProVerif [288, Sec. 3.1.4], ensures that the consumer listeners only work on update rounds intended for their vehicle.

9.7.3.9 Unbounded Number of Processes in ProVerif and Considerations

Using unbounded processes running concurrently in ProVerif [288] means that our update rounds can also occur concurrently in our model. However, [308] notes that this may not accurately represent reality, where update rounds occur sequentially for a given vehicle. The strict sequential update rounds could be achieved by allowing a vehicle to update only once. Although this would simplify the solution, it would also trivialize the problem, as the adversary would not be able to replay messages to the same vehicle since there would be no other update rounds to target. We believe that the use of unbounded concurrent processes is crucial to our proof. We argue that the set of all sequential schedules is a subset of the set of all concurrent schedules. Therefore, if our proof holds for all concurrent schedules, it will also hold for all sequential schedules.

9.7.3.10 Well-Known Addresses

We add a communication channel for each pair of entities communicating in a task. This allows us to separate the different peer-to-peer sessions inside an update round from each other. Without multiple channels, a message from Alice to Bob could end up at Charlie, that is, the channels simulate the well-known addresses discussed in Section 9.3.1. Specifically, channels alone simulate well-known addresses for the producer and the software repository. Meanwhile, for the consumer entities that belong to a vehicle, the VIN is also needed to simulate well-known addresses. The reason is that the VIN separates the multiple vehicles considered in our system setup (see Section 9.7.3.1).

9.7.4 ProVerif Libraries

ProVerif offers a method to organize frequently used functions and macros into a library file, allowing them to be imported into other files to minimize redundant code [288, Sec. 6.6]. In addition, the libraries ensure that the data structures appearing in multiple proofs are modeled consistently. We have developed libraries for implementing cryptographic primitives, helper functions for setting up cryptographic materials, and common message types.

9.7.5 Correctness Proof for Intra-Round Uniqueness

Lemma 9.7.1. *Consider an entity E and a handling $e(r,\ d,\ \ell)$, E executes $e(r,\ d,\ \ell)$ at most once.*

Proof. Consider an entity, E, and a handling event, $e(r,\ d,\ \ell)$, which we denote e for brevity. To execute e, entity E must be in the update round r and have previously generated the cryptographic material d. Then, there can only be two cases: either the generation of d depends on some cryptographic materials sent to E in messages, m', or E generates d entirely from scratch. For the sake of simplicity, we assume that there is only one such message. Note that similar arguments are held when multiple messages are held.

In the first case, m' must contain $(r,\ d')$ where all d' were used to generate d. To execute the event e more than once, E must have received multiple versions of message m' containing $(r,\ d')$. However, this is impossible because the entities never process duplicate messages, according to the assumption in Section 9.3.4, which specifies that entities omit any message already in their logs.

In the second case, d is generated from scratch. As we assume perfect cryptography and, hereby, randomness is based on a random oracle, two generated materials cannot be identical. Therefore, executing e with the same d multiple times is impossible.

Thus, executing e multiple times in both cases is impossible. In other words, E executes e at most once. □

From the Lemma 9.7.1, we derive that the *Intra-Round Uniqueness* requirement always holds.

Corollary 9.7.1.1. *The System-level Requirement 9.3.5.4 holds for all tasks.*

9.7.6 Correctness Proof for Termination

Lemma 9.7.2. *All entity executions terminate eventually. Each termination is either timely or late.*

Proof. Consider an update round; if an entity runs its *halt* task for this update round before the expiration time, it terminates timely by the definition of termination in Section 9.3.5. Otherwise, it fails to run the halt task before the expiration time, and by assumption (see Section 9.5.0.2), the entity halts and terminates late. Therefore, all entity executions always terminate and each termination is either timely or late. □

From the Lemma 9.7.2, we derive that the *Termination* requirement always holds because all update rounds must always terminate if all entities always terminate.

Corollary 9.7.2.1. *The System-level Requirement 9.3.5.6 holds for all tasks.*

9.8 Conclusions

Our work scrutinizes UniSUF's requirements and our research questions (see Section 9.1.1). To validate UniSUF's requirements and architecture, we developed a formal model using ProVerif to ensure that the ProVerif program satisfies the specified requirements and the technological assumptions that UniSUF relies on. Furthermore, we divided the UniSUF update process into smaller, more manageable sub-problems. We analyzed and created specific requirements for each sub-problem so that the requirements of the sub-problems together fulfill the System-level Requirements of UniSUF (see Section 9.3.5).

Our verification results show that our symbolic execution of UniSUF in ProVerif fulfils all requirements. The system-level requirements established for UniSUF collectively address the research questions posed in Section 9.1.1. The *Confidential Secrets* requirement (System-level Requirement 9.3.5.1) ensures that the system's secrets, such as cryptographic keys and disseminated software, are not exposed to the adversary, addressing **RQ1**. The *Integrity of Cryptographic Materials* requirement (System-level Requirement 9.3.5.2) guarantees that the software obtained and used by UniSUF originates from the right source and has not been modified by any other entity, addressing **RQ2**. The *Inter-Round Uniqueness* and *Intra-Round Uniqueness* (System-level Requirement 9.3.5.3 and System-level Requirement 9.3.5.4) collaboratively prevent the use of obsolete software versions and the replay of cryptographic materials within and between update rounds. This guarantees that UniSUF always performs software updates with the correct up-to-date versions, thus addressing **RQ3**. The *Integrity of Handling Events* (System-level Requirement 9.3.5.5) ensures the operations in the software update process follow the order specified by UniSUF, thus addressing **RQ4**. The last requirement, *Termination* (System-level Requirement 9.3.5.6), ensures that the update process always terminates, addressing **RQ5**.

Although our work proves the UniSUF model in ProVerif to be secure, it does not necessarily guarantee the security of a real-world implementation of UniSUF. There is an inherent discrepancy between the latter and our formal model. Namely, formal models are intended to be complete, e.g., nothing outside the model's specification can occur, such as compromised components. However, in real-world deployments, implementation errors, and unexpected events, such as evolving attacker capabilities, can affect the system security. Thus, important challenges remain in the area.

This does not imply that our formal verification has failed to establish meaningful security guarantees. On the contrary, our work has rigorously demonstrated the security properties of UniSUF in the formal model. Our work establishes the provability of UniSUF security, which can be a starting point for real-world implementations of UniSUF.

However, the security assurances provided by our model do not automatically transfer to a real-world implementation. Verifying the correctness of an actual UniSUF implementation requires a substantially different and more comprehensive analysis that goes beyond the scope of our work. Thus, bridging the gap between the formal model and a real-world implementation remains an open challenge and requires further investigation.

9.9 Appendix A: Implementations in ProVerif

9.9.1 Cryptographic Primitives

```
type key.

(* Helper to convert key to bitstring and back so that it can be encrypted*)
fun key2bits(key): bitstring [data, typeConverter].
fun bits2key(bitstring): key [data, typeConverter].

(* Authenticated symmetric encryption *)
fun authSenc(bitstring, key): bitstring.
reduc forall m: bitstring, k: key; authSdec(authSenc(m, k), k) = m.

(* Unathenticated symmetric encryption *)
fun senc(bitstring, key): bitstring.
fun sdec(bitstring, key): bitstring.
equation forall m: bitstring, k:key; sdec(senc(m,k), k) = m.
equation forall m: bitstring, k:key; senc(sdec(m,k), k) = m.

(* Asymmetric encryption *)
type skey.
type pkey.

fun pk(skey): pkey.
fun aenc(bitstring, pkey): bitstring.
reduc forall m: bitstring, k: skey; adec(aenc(m, pk(k)), k) = m.

fun hash(bitstring): bitstring.

type sskey.
type spkey.
fun spk(sskey): spkey.
fun sgnHash(bitstring, sskey): bitstring.
fun createSgn(bitstring, bitstring): bitstring.

reduc forall m: bitstring, k: sskey; getMess(createSgn(m, sgnHash(hash(m),
↪  k))) = m.
reduc forall m: bitstring, k: sskey; getHash(createSgn(m, sgnHash(hash(m),
↪  k)), spk(k)) = hash(m).
reduc forall m: bitstring, k: sskey; validateSgn(createSgn(m, sgnHash(hash(m),
↪  k)), spk(k)) = m.

fun sgn(bitstring, sskey): bitstring.
equation forall m: bitstring, ssk: sskey;
    createSgn(m, sgnHash(hash(m), ssk)) = sgn(m, ssk).

(* Certificate stuff *)
(*
    A ceritificate can be seen as an extension to a public key, that provides
    ↪  a
    signature of the public key signed by the issuer's secret key.
*)
type cert.

fun spkey2bits(spkey): bitstring [data, typeConverter].
fun bits2spkey(bitstring): spkey [data, typeConverter].

fun createCert(spkey, sskey): cert.
reduc forall holderSpk: spkey, issuerSsk: sskey;
    validateCert(createCert(holderSpk, issuerSsk), spk(issuerSsk)) =
    ↪  (holderSpk, spk(issuerSsk)).
reduc forall holderSpk: spkey, issuerSsk: sskey;
    getCert(createCert(holderSpk, issuerSsk)) = (holderSpk, spk(issuerSsk)).

(* Other type converters *)

(* In principle, these two type of secret keys should be interchangeable *)
fun skey2sskey(skey): sskey [data, typeConverter].
fun sskey2skey(sskey): skey [data, typeConverter].
(* similarily, here should also be interchangeable *)
fun pkey2spkey(pkey): spkey [data, typeConverter].
fun spkey2pkey(spkey): pkey [data, typeConverter].
```

Listing 9.14: ProVerif library file `cryptography.pvl` that contains all our cryptographic primitives.

9.9.2 Utilities

```
(* -lib cryptography.pvl *)

fun setupVso(bitstring): bitstring[data].
reduc forall vin: bitstring; getVin(setupVso(vin)) = vin.

letfun setupVsoSgn(cdaSsk: sskey, vin: bitstring) =
    sgn(setupVso(vin), cdaSsk).

(* Authenticated encryption of key k, via authEncKey*)
letfun authSencKey (k: key, encK: key) =
    authSenc(key2bits(k), encK).

(*
    Generates a new signed key manifest, and returns the session key along
    ↪  with the signed manifest
*)
letfun genKeyManifestSgn(sessionK: key, vin: bitstring, expirationTime:
↪  bitstring, vehiclePk: pkey, sgnSsk: sskey) =
    let kEnc = aenc(key2bits(sessionK), vehiclePk) in
    sgn((kEnc, (vin, expirationTime)), sgnSsk).

(*
fun vuupSgn2vuupUrl(bitstring): bitstring.
reduc forall vuupSgn: bitstring; vuupUrl2vuupSgn(vuupSgn2vuupUrl(vuupSgn)) =
↪  vuupSgn.

letfun setupVuupUrlSgn(vuupSgn: bitstring, vcmSsk: sskey) =
    sgn(vuupSgn2vuupUrl(vuupSgn), vcmSsk).
*)
letfun setupVuupUrlSgn(vcmSsk: sskey) =
    new vuupUrl: bitstring;
    (sgn(vuupUrl, vcmSsk), vuupUrl).

letfun setupDkmSgn(vin: bitstring, expirationTime: bitstring, vehiclePk: pkey,
↪  pdaSsk: sskey) =
    new dkmK: key;
    let dkmSgn = genKeyManifestSgn(dkmK, vin, expirationTime, vehiclePk,
    ↪  pdaSsk) in
    (dkmK, dkmSgn).

letfun setupIkmSgn(vin: bitstring, expirationTime: bitstring, vehiclePk: pkey,
↪  piaSsk: sskey) =
    new ikmK: key;
    let ikmSgn = genKeyManifestSgn(ikmK, vin, expirationTime, vehiclePk,
    ↪  piaSsk) in
    (ikmK, ikmSgn).


fun createDownloadInstructions(bitstring): bitstring.
fun createInstallationInstructions(bitstring, bitstring, bitstring, cert):
↪  bitstring.
(*
    Reducer does not include the signed software list as we do not want to be
    ↪  able to compute back to it
    TODO: ASK KIM IF THIS IS ACTUALLY THE CASE
    Same with no reducer for dowload instructions, as we do not wnat to be
    ↪  able to recompute the software list
*)
reduc forall softwareListSgn: bitstring, skaSgn: bitstring, mkmSgn:bitstring,
↪  psaCert: cert;

    ↪  unpackInstallationInstructions(createInstallationInstructions(softwareListSgn,
    ↪  skaSgn, mkmSgn, psaCert)) = (skaSgn, mkmSgn, psaCert).


letfun setupSoftwareEncapsulated(supplierSsk: sskey, skaSoftwareK: key,
↪  vcmSsk: sskey) =
    new software: bitstring;
    sgn(senc(sgn(software, supplierSsk), skaSoftwareK), vcmSsk).

letfun setupDownloadInstructionsEncSgn(softwareListSgn: bitstring, dkmK: key,
↪  pdaSsk: sskey) =
    let downloadInstructions = createDownloadInstructions(softwareListSgn) in
    sgn(senc(downloadInstructions, dkmK), pdaSsk).
```

```

letfun setupMkm(vin: bitstring, expirationTime: bitstring, vehiclePk: pkey) =
    (*Setting up MKM*)
    new mkmEcuK: key;
    new mkmSoftwareK: key;
    let mkmEcu = (aenc(key2bits(mkmEcuK), vehiclePk),
    (vin, expirationTime)) in
    let mkmSoftware = (aenc(key2bits(mkmSoftwareK), vehiclePk), (vin,
    ↪  expirationTime)) in
    let mkm = (mkmEcu, mkmSoftware) in
    (mkmEcuK, mkmSoftwareK, mkm).

letfun setupMkmSgn(vin: bitstring, expirationTime: bitstring, vehiclePk: pkey,
↪  psaSsk: sskey) =
    let (mkmEcuK: key, mkmSoftwareK: key, mkm:bitstring) = setupMkm(vin,
    ↪  expirationTime, vehiclePk) in
    (mkmEcuK, mkmSoftwareK, sgn(mkm, psaSsk)).

letfun setupSkaSoftwareK () =
    new skaSoftwareK: key;
    (skaSoftwareK).

letfun setupSka(mkmEcuK: key, mkmSoftwareK: key) =
    (* Setting up SKA, assuming only one ECU and only one sofware file
    ↪  therefore only
       one key per sub-array.

       In reality, e.g.: skaEcu = [authSenc(skaEcuK1, mkmEcuK), ...,
       ↪  authSenc(skaEcuKn, mkmEcuK)]
       However, here we have simplified it such that skaEcu =
       ↪  authSenc(skaEcuK, mkmEcuK)
    *)
    new skaEcuK: key;
    let skaEcu = authSencKey(skaEcuK, mkmEcuK) in
    let skaSoftwareK = setupSkaSoftwareK() in
    (*new skaSoftwareK: key;*)

    let skaSoftware = authSencKey(skaSoftwareK, mkmSoftwareK) in
    (skaEcuK, skaSoftwareK, (skaEcu, skaSoftware)).

letfun setupSkaSgn (mkmEcuK: key, mkmSoftwareK: key, psaSsk: sskey) =
    let (skaEcuK: key, skaSoftwareK: key, ska: bitstring) = setupSka(mkmEcuK,
    ↪  mkmSoftwareK) in
    (skaEcuK, skaSoftwareK, sgn(ska, psaSsk)).

fun setupSoftwareList(bitstring, bitstring): bitstring.

letfun setupSoftwareListSgn(vcmSsk: sskey) =
    (*Setting up softwareList*)
    new vinData: bitstring;
    new softwareVersions: bitstring;
    sgn(setupSoftwareList(vinData, softwareVersions), vcmSsk).

letfun setupInstallationInstructionsEncSgn(softwareListSgn: bitstring, skaSgn:
↪  bitstring, mkmSgn: bitstring, psaCert: cert, ikmK: key, piaSsk: sskey) =
    let installationInstructions: bitstring =
    ↪  createInstallationInstructions(softwareListSgn, skaSgn, mkmSgn,
    ↪  psaCert) in
    sgn(senc(installationInstructions, ikmK), piaSsk).

letfun setupVuupContent (vin: bitstring, expirationTime: bitstring, pdaCert:
↪  cert, piaCert: cert, downloadInstructionsEncSgn: bitstring, dkmSgn:
↪  bitstring, installationInstructionsEncSgn: bitstring, ikmSgn: bitstring) =
    (vin, expirationTime, (pdaCert, piaCert), downloadInstructionsEncSgn,
    ↪  dkmSgn, installationInstructionsEncSgn, ikmSgn).

letfun setupVuupSgn (vin: bitstring, expirationTime: bitstring, pdaCert: cert,
↪  piaCert: cert, downloadInstructionsEncSgn: bitstring, dkmSgn: bitstring,
↪  installationInstructionsEncSgn: bitstring, ikmSgn: bitstring, vcmSsk:
↪  sskey, vcmCert: cert) =
    (vcmCert, sgn(setupVuupContent(vin, expirationTime, pdaCert, piaCert,
    ↪  downloadInstructionsEncSgn, dkmSgn, installationInstructionsEncSgn,
    ↪  ikmSgn), vcmSsk)).

```

Listing 9.15: ProVerif library file `uniSufHelpers.pvl` that contains all helper methods for setting up the cryptographic materials in Section 9.4.1.

9.9.3 Message Types

```
type vcmInitSuccess.
type pdaSlSuccess.
type piaSlSuccess.
type psaSlSuccess.
type dkmSuccess.
type ikmSuccess.
type mkmSuccess.
type vuupUrlSuccess.

type vsoRequest.
type dkmKeyRequest.
type ikmKeyRequest.
type vehicleCertRequest.
type vuupUrlRequest.
type vuupRequest.
type vuupSuccess.
type softwareSuccess.
type softwareRequest.
type signatureRequest.
type skaSgnRequest.
type mkmSgnRequest.
type cmsCryptoRequest.
```

Listing 9.16: ProVerif library file `uniSufMessageTypes.pvl` that contains all the message types used in our implementation.

Bibliography

[1] B. Canis and D. R. Peterman, “UN Regulations on Cybersecurity and Software Updates to pave the way for mass roll out of connected vehicles,” https://unece.org/sustainable-development/press/un-regulations-cybersecurity-and-software-updates-pave-way-mass-roll, 2020, accessed: 2024-08-26.

[2] Volvo Cars - Global Newsroom, “Heritage - Volvo OV4,” https://www.media.volvocars.com/global/en-gb/heritagemodels/volvo-ov4/227014, 2024, accessed: 2024-09-03.

[3] Volvo Cars, “Volvo Cars - Global Newsroom - Copyright Volvo Cars,” https://www.media.volvocars.com/, 2021, accessed: 2023-12-17.

[4] K. Strandberg, N. Nowdehi, and T. Olovsson, “A systematic literature review on automotive digital forensics: Challenges, technical solutions and data collection,” *IEEE Transactions on Intelligent Vehicles*, pp. 1–19, 2022.

[5] “ISO 26262:2011 Road Vehicles – Functional Safety,” International Organization for Standardization (ISO), Standard, 2011.

[6] “ISO/SAE 21434 Road Vehicles – Cybersecurity Engineering,” International Organization for Standardization (ISO), Standard, 2020.

[7] United Nations Economic Commission for Europe, “Regulation document title,” https://unece.org/sites/default/files/2023-10/R160E.pdf, 2023, accessed: 2024-03-21,.

[8] United Nations, “UN Regulation No. 155,” https://unece.org/sites/default/files/2021-03/R155e.pdf, 2022, accessed: 2022-06-08.

[9] Proton Technologies AG, “Complete guide to GDPR compliance,” https://gdpr.eu/, 2020, accessed: 2020-11-17.

[10] United Nations Economic Commission for Europe (UNECE), “UN Regulation No. 156 - Software update and software update management system,” https://unece.org/transport/documents/2021/03/standards/un-regulation-no-156-software-update-and-software-update, 2021, accessed: 2024-03-21.

[11] International Organization for Standardization, "Road vehicles — Software update engineering," https://www.iso.org/standard/77796.html, 2021, accessed: 2021-06-02.

[12] National Institute of Standards and Technology, "Approaches for Federal Agencies to Use the Cybersecurity Framework," https://doi.org/10.6028/NIST.IR.8170-upd, 2020, accessed: 2021-11-24.

[13] T. Rosenstatter, K. Strandberg, R. Jolak, R. Scandariato, and T. Olovsson, "REMIND: A framework for the resilient design of automotive systems," *IEEE Secure Development*, 2020, in press.

[14] Microsoft Corporation, "The STRIDE Threat Model," https://docs.microsoft.com/en-us/previous-versions/commerce-server/ee823878(v=cs.20), 2009, accessed: 2021-11-22.

[15] United Nations Economic Commission for Europe (UNECE), "Un regulation no. 156 - software update and software update management system," 2022.

[16] "Microsoft Security Development Lifecycle Practices," https://www.microsoft.com/en-us/securityengineering/sdl/practices, accessed: April 16, 2024.

[17] "AUTOSAR: Automotive open system architecture," https://www.autosar.org, accessed: April 8, 2024.

[18] K. Strandberg, D. K. Oka, and T. Olovsson, "UniSUF: a unified software update framework for vehicles utilizing isolation techniques and trusted execution environments," 2021.

[19] K. Strandberg, U. Arnljung, T. Olovsson, and D. K. Oka, "Secure vehicle software updates: Requirements for a reference architecture," in *2023 IEEE 97th Vehicular Technology Conference*, 2023.

[20] K. Strandberg, U. Arnljung, and T. Olovsson, "The automotive blackbox: Towards a standardization of automotive digital forensics," in *2023 IEEE International Workshop on Information Forensics and Security (WIFS)*, 2023, pp. 1–6.

[21] "ISO14229, Road vehicles - Unified Diagnostic Services (UDS)," International Organization for Standardization (ISO), Tech. Rep., 2020.

[22] C. Miller and C. Valasek, "Remote exploitation of an unaltered passenger vehicle," *Black Hat USA*, 2015.

[23] S. Checkoway, D. McCoy, B. Kantor, D. Anderson, H. Shacham *et al.*, "Comprehensive experimental analyses of automotive attack surfaces," in *USENIX Security Symposium.* San Francisco, 2011.

[24] K. Koscher, A. Czeskis, F. Roesner, S. Patel, T. Kohno *et al.*, "Experimental security analysis of a modern automobile," in *2010 IEEE Symposium on Security and Privacy*, 2010, pp. 447–462.

[25] Reuter, “Uber, distracted backup driver cited by NTSB in fatal self-driving crash,” https://www.reuters.com/article/us-uber-crash/ntsb-cites-uber-distracted-backup-driver-in-fatal-self-driving-crash-idUSKBN1XT2IL, 2019, accessed: 2021-02-02.

[26] Reuters, “EXCLUSIVE Dutch forensic lab says it has decoded Tesla’s driving data,” https://www.reuters.com/business/autos-transportation/dutch-forensic-lab-says-it-has-decoded-teslas-driving-data-2021-10-21/, 2022, accessed: 2022-06-10.

[27] Andrew J. Hawkins, “Tesla’s Autopilot and Full Self-Driving linked to hundreds of crashes, dozens of deaths,” visited on 2024-10-22. [Online]. Available: https://www.theverge.com/2024/4/26/24141361/tesla-autopilot-fsd-nhtsa-investigation-report-crash-death

[28] “IEEE Standard for Data Storage Systems for Automated Driving,” *IEEE Std 1616.1-2023*, pp. 1–43, 2023.

[29] K. Strandberg, T. Olovsson, and E. Jonsson, “Securing the connected car: A security-enhancement methodology,” *IEEE Vehicular Technology Magazine*, vol. 13, no. 1, pp. 56–65, 2018.

[30] K. Strandberg, T. Rosenstatter, R. Jolak, N. Nowdehi, and T. Olovsson, “Resilient shield: Reinforcing the resilience of vehicles against security threats,” in *2021 IEEE 93rd Vehicular Technology Conference (VTC2021-Spring)*, 2021, pp. 1–7.

[31] K. Strandberg, D. Kengo Oka, and T. Olovsson, “Unisuf: a unified software update framework for vehicles utilizing isolation techniques and trusted execution environments,” in *19th escar Europe : The World’s Leading Automotive Cyber Security Conference*, 2021, pp. 86–100.

[32] T. Llansó and M. McNeil, “Estimating software vulnerability counts in the context of cyber risk assessments,” in *HICSS*, 2018.

[33] M. Mirakhorli, D. Garcia, S. Dillon, K. Laporte, M. Morrison, H. Lu, V. Koscinski, and C. Enoch, “A landscape study of open source and proprietary tools for software bill of materials (sbom),” *arXiv preprint arXiv:2402.11151*, 2024. [Online]. Available: https://doi.org/10.48550/arXiv.2402.11151

[34] The Industrial Control Systems Cyber Emergency Response Team (ISC-CERT), “Alert (ICS-ALERT-15-203-01),” https://us-cert.cisa.gov/ics/alerts/ICS-ALERT-15-203-01, 2015, accessed: 2021-11-22.

[35] R. Baldwin, “OwnStar car hacker can remotely unlock BMWs, Benz and Chrysler, engadget),” http://www.engadget.com/2015/08/13/ownstar-hack/, 2015, accessed: 2021-11-22.

[36] A. Ruddle, D. Ward, B. Weyl, S. Idrees, Y. Roudier, M. Friedewald, T. Leimbach *et al.*, “Deliverable D2.3: Security requirements for automotive on-board networks based on dark-side scenarios,” E-safety vehicle intrusion protected applications (EVITA), Deliverable, 2009.

[37] M. Islam, A. Lautenbach, C. Sandberg, and T. Olovsson, “A risk assessment framework for automotive embedded systems,” in *Proceedings of the 2nd ACM International Workshop on Cyber-Physical System Security - CPSS 16*. Association for Computing Machinery (ACM), 2016.

[38] M. Rosenquist and T. Casey, “Prioritizing information security risks with threat agent risk assessment (tara),” 12 2009.

[39] K. Strandberg, “Avoiding Vulnerabilities in Connected Cars,” https://publications.lib.chalmers.se/records/fulltext/238172/238172.pdf, 2016, accessed: 2021-11-22.

[40] S. Harris, *CISSP All-in-One Exam Guide, Seventh Edition*. McGraw-Hill Education, 2016.

[41] OffSec Services Limited, “Kali Tools,” https://www.kali.org/tools/, 2021, accessed: 2021-11-22.

[42] Hak5, “Kali Tools,” https://www.wifipineapple.com, 2021, accessed: 2021-11-22.

[43] K.-T. Cho and K. G. Shin, “Error handling of in-vehicle networks makes them vulnerable,” in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS ’16. New York, NY, USA: Association for Computing Machinery, 2016, p. 1044–1055. [Online]. Available: https://doi.org/10.1145/2976749.2978302

[44] Vector, “Testing ECUs and Networks with CANoe,” https://www.vector.com/se/en-se/products/products-a-z/software/canoe/, 2021, accessed: 2021-11-22.

[45] “NISTIR 7628 Rev 1 – Guidelines for smart grid cybersecurity,” National Institute of Standards and Technology, Tech. Rep., Sep. 2014. [Online]. Available: https://doi.org/10.6028/NIST.IR.7628r1

[46] “The Guidelines on Cyber Security Onboard Ships,” BIMCO, CLIA, ICS, INTERCARGO, INTERMANAGER, INTERTANKO, IUMI, OCIMF and WORLD SHIPPING COUNCIL, Tech. Rep., Dec. 2017. [Online]. Available: https://www.ics-shipping.org/docs/default-source/resources/safety-security-and-operations/guidelines-on-cyber-security-onboard-ships.pdf

[47] “Cyber Security and Resilience of smart cars,” The European Union Agency for Network and Information Security (ENISA), Tech. Rep., 2016.

[48] “SAE J3061: Cybersecurity Guidebook for Cyber-Physical Vehicle Systems,” SAE International, Standard, 2016.

[49] UNECE, “TFCS-09-14 Draft Recommendation on Cyber Security of the Task Force on CyberSecurity and Over-the-air issues of UNECE WP.29 IWG ITS/AD,” 2017.

[50] J.-C. Laprie, "From dependability to resilience," in *38th IEEE/IFIP Int. Conf. On Dependable Systems and Networks*, 2008, pp. G8–G9.

[51] J. P. Sterbenz, D. Hutchison, E. K. Çetinkaya, A. Jabbar, J. P. Rohrer, M. Schöller, and P. Smith, "Resilience and survivability in communication networks: Strategies, principles, and survey of disciplines," *Computer Networks*, vol. 54, no. 8, pp. 1245 – 1265, 2010, resilient and Survivable networks. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1389128610000824

[52] O. Andersson, "The Car - A Computer on Wheels," URL: https://www.icse2018.org/getImage/orig/The+Car+%E2%80%93+computer+on+wheels.pdf, May 2018, visited on 2020-05-21.

[53] V. Chang, M. Ramachandran, Y. Yao, Y.-H. Kuo, and C.-S. Li, "A resiliency framework for an enterprise cloud," *International Journal of Information Management*, vol. 36, no. 1, pp. 155 – 166, 2016. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S026840121500095X

[54] S. Hukerikar and C. Engelmann, "Resilience design patterns: A structured approach to resilience at extreme scale," *arXiv preprint arXiv:1708.07422*, 2017.

[55] R. Ross, V. Pillitteri, R. Graubart, D. Bodeau, and R. McQuaid, "Developing cyber resilient systems:: a systems security engineering approach," National Institute of Standards and Technology, Gaithersburg, MD, Tech. Rep. NIST SP 800-160v2, Nov. 2019. [Online]. Available: https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-160v2.pdf

[56] D. Ratasich, F. Khalid, F. Geissler, R. Grosu, M. Shafique, and E. Bartocci, "A Roadmap Toward the Resilient Internet of Things for Cyber-Physical Systems," *IEEE Access*, vol. 7, pp. 13 260–13 283, 2019.

[57] J. P. Sterbenz, D. Hutchison, E. K. Çetinkaya, A. Jabbar, J. P. Rohrer, M. Schöller, and P. Smith, "Redundancy, diversity, and connectivity to achieve multilevel network resilience, survivability, and disruption tolerance invited paper," *Telecommunication Systems*, vol. 56, no. 1, pp. 17–31, 2014.

[58] M. Segovia, A. R. Cavalli, N. Cuppens, and J. Garcia-Alfaro, "A study on mitigation techniques for scada-driven cyber-physical systems (position paper)," in *Foundations and Practice of Security*, N. Zincir-Heywood, G. Bonfante, M. Debbabi, and J. Garcia-Alfaro, Eds. Cham: Springer International Publishing, 2019, pp. 257–264.

[59] Z. Bakhshi, G. Rodriguez-Navas, and H. Hansson, "Dependable Fog Computing: A Systematic Literature Review," in *2019 45th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*, 2019, pp. 395–403.

[60] I. Egwutuoha, D. Levy, B. Selic, and S. Chen, "A survey of fault tolerance mechanisms and checkpoint/restart implementations for high performance

computing systems," *The Journal of Supercomputing*, vol. 65, no. 3, pp. 1302–1326, 2013.

[61] P. Kumari and P. Kaur, "A survey of fault tolerance in cloud computing," *Journal of King Saud University - Computer and Information Sciences*, 2018. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1319157818306438

[62] M. A. Mukwevho and T. Celik, "Toward a smart cloud: A review of fault-tolerance methods in cloud systems," *IEEE Transactions on Services Computing*, pp. 1–1, 2018.

[63] Vidar Slåtten, Peter Herrmann, and Frank Alexander Kraemer, "Chapter 4 - model-driven engineering of reliable fault-tolerant systems—a state-of-the-art survey," in *Advances in Computers*, ser. Advances in Computers, A. Memon, Ed. Elsevier, 2013, vol. 91, pp. 119 – 205. [Online]. Available: http://www.sciencedirect.com/science/article/pii/B9780124080898000045

[64] D. Wanner, A. Trigell, L. Drugge, and J. Jerrelind, "Survey on fault-tolerant vehicle design," *World Electric Vehicle Journal*, vol. 5, no. 2, p. 598–609, Jun 2012. [Online]. Available: http://dx.doi.org/10.3390/wevj5020598

[65] E. Bartocci and Y. Falcone, *Lectures on Runtime Verification: Introductory and Advanced Topics.* Springer, Cham, 2018, vol. 10457.

[66] D. Heffernan, C. Macnamee, and P. Fogarty, "Runtime verification monitoring for automotive embedded systems using the ISO 26262 functional safety standard as a guide for the definition of the monitored properties," *IET Software*, vol. 8, no. 5, pp. 193–203, 2014.

[67] M. Müter, A. Groll, and F. C. Freiling, "A structured approach to anomaly detection for in-vehicle networks," in *2010 Sixth International Conference on Information Assurance and Security*, 2010, pp. 92–98.

[68] N. Nowdehi, W. Aoudi, M. Almgren, and T. Olovsson, "CASAD: CAN-Aware Stealthy-Attack Detection for In-Vehicle Networks," *arXiv preprint arXiv:1909.08407*, 2019.

[69] M. Hanselmann, T. Strauss, K. Dormann, and H. Ulmer, "Canet: An unsupervised intrusion detection system for high dimensional can bus data," *IEEE Access*, vol. 8, pp. 58 194–58 205, 2020.

[70] M. Müter and N. Asaj, "Entropy-based anomaly detection for in-vehicle networks," in *2011 IEEE Intelligent Vehicles Symposium (IV)*, 2011, pp. 1110–1115.

[71] K.-T. Cho and K. G. Shin, "Viden: Attacker identification on in-vehicle networks," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '17. New York, NY, USA: Association for Computing Machinery, 2017, p. 1109–1123. [Online]. Available: https://doi.org/10.1145/3133956.3134001

[72] M. Husák, J. Komárková, E. Bou-Harb, and P. Čeleda, "Survey of attack projection, prediction, and forecasting in cyber security," *IEEE Communications Surveys Tutorials*, vol. 21, no. 1, pp. 640–660, 2019.

[73] H. Kopetz, *Real-time systems: design principles for distributed embedded applications.* Springer Science & Business Media, 2011.

[74] F. Gustafsson, "Particle filter theory and practice with positioning applications," *IEEE Aerospace and Electronic Systems Magazine*, vol. 25, no. 7, pp. 53–82, 2010.

[75] L. Chen and A. Avizienis, "N-version programming: A fault-tolerance approach to reliability of software operation," in *Proc. 8th IEEE Int. Symp. on Fault-Tolerant Computing (FTCS-8)*, vol. 1, 1978, pp. 3–9.

[76] J.-C. Laprie, J. Arlat, C. Beounes, and K. Kanoun, "Definition and analysis of hardware-and software-fault-tolerant architectures," *Computer*, vol. 23, no. 7, pp. 39–51, 1990.

[77] B. Cox, D. Evans, A. Filipi, J. Rowanhill, W. Hu, J. Davidson, J. Knight, A. Nguyen-Tuong, and J. Hiser, "N-Variant Systems: A Secretless Framework for Security through Diversity," in *15th USENIX Security Symposium*, 2006, pp. 105–120.

[78] A. Höller, T. Rauter, J. Iber, and C. Kreiner, "Towards dynamic software diversity for resilient redundant embedded systems," in *Software Engineering for Resilient Systems*, A. Fantechi and P. Pelliccione, Eds. Cham: Springer International Publishing, 2015, pp. 16–30.

[79] T. Dagan, Y. Montvelisky, M. Marchetti, D. Stabili, M. Colajanni, and A. Wool, "Vehicle safe-mode, concept to practice limp-mode in the service of cybersecurity," *SAE Int. J. Transp. Cyber. & Privacy 2 (2)*, feb 2020.

[80] T. Ishigooka, S. Otsuka, K. Serizawa, R. Tsuchiya, and F. Narisawa, "Graceful degradation design process for autonomous driving system," in *Computer Safety, Reliability, and Security*, A. Romanovsky, E. Troubitsyna, and F. Bitsch, Eds. Cham: Springer International Publishing, 2019, pp. 19–34.

[81] A. Reschka, G. Bagschik, S. Ulbrich, M. Nolte, and M. Maurer, "Ability and skill graphs for system modeling, online monitoring, and decision support for vehicle guidance systems," in *2015 IEEE Intelligent Vehicles Symposium (IV)*, 2015, pp. 933–939.

[82] J. Rubio-Hernan, R. Sahay, L. De Cicco, and J. Garcia-Alfaro, "Cyber-physical architecture assisted by programmable networking," *Internet Technology Letters*, vol. 1, no. 4, p. e44, 2018. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/itl2.44

[83] Z. Jiang, N. C. Audsley, and P. Dong, "BlueVisor: A Scalable Real-Time Hardware Hypervisor for Many-Core Embedded Systems," in *2018 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, 2018, pp. 75–84.

[84] P. Alho and J. Mattila, “Service-oriented approach to fault tolerance in cpss,” *Journal of Systems and Software*, vol. 105, pp. 1 – 17, 2015. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0164121215000643

[85] M. Wu, H. Zeng, C. Wang, and H. Yu, “INVITED: Safety guard: Runtime enforcement for safety-critical cyber-physical systems,” in *2017 54th ACM/EDAC/IEEE Design Automation Conference (DAC)*, 2017, pp. 1–6.

[86] L. F. Cómbita, J. Giraldo, A. A. Cárdenas, and N. Quijano, “Response and reconfiguration of cyber-physical control systems: A survey,” in *2015 IEEE 2nd Colombian Conference on Automatic Control (CCAC)*, 2015, pp. 1–6.

[87] Y. Zhang and J. Jiang, “Bibliographical review on reconfigurable fault-tolerant control systems,” *Annual Reviews in Control*, vol. 32, no. 2, pp. 229 – 252, 2008. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1367578808000345

[88] M. Möstl, J. Schlatow, R. Ernst, N. Dutt, A. Nassar, A. Rahmani, F. J. Kurdahi, T. Wild, A. Sadighi, and A. Herkersdorf, “Platform-Centric Self-Awareness as a Key Enabler for Controlling Changes in CPS,” *Proceedings of the IEEE*, vol. 106, no. 9, pp. 1543–1567, 2018.

[89] T. D. Nya, S. C. Stilkerich, and C. Siemers, “Self-aware and self-expressive driven fault tolerance for embedded systems,” in *2014 IEEE Symposium on Intelligent Embedded Systems (IES)*, Dec 2014, pp. 27–33.

[90] S. Zeadally, T. Sanislav, and G. D. Mois, “Self-Adaptation Techniques in Cyber-Physical Systems (CPSs),” *IEEE Access*, vol. 7, pp. 171 126–171 139, 2019.

[91] D. Weyns, *Software Engineering of Self-adaptive Systems.* Cham: Springer International Publishing, 2019, pp. 399–443. [Online]. Available: https://doi.org/10.1007/978-3-030-00262-6_11

[92] H. Zhang, B. Huang, P. Zhang, and H. Ju, “A New SoS Engineering Philosophy - Vitality Theory,” in *2019 14th Annual Conference System of Systems Engineering (SoSE)*, May 2019, pp. 19–24.

[93] G. Vachtsevanos, B. Lee, S. Oh, and M. Balchanos, “Resilient design and operation of cyber physical systems with emphasis on unmanned autonomous systems,” *Journal of Intelligent & Robotic Systems*, vol. 91, no. 1, pp. 59–83, 2018.

[94] R. de Lemos, H. Giese, H. A. Müller, and M. Shaw et al., *Software Engineering for Self-Adaptive Systems: A Second Research Roadmap.* Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 1–32. [Online]. Available: https://doi.org/10.1007/978-3-642-35813-5_1

[95] T. Hoppe, S. Kiltz, and J. Dittmann, “Security threats to automotive CAN networks—Practical examples and selected short-term

countermeasures," *Reliability Engineering & System Safety*, vol. 96, no. 1, pp. 11 – 25, 2011, special Issue on Safecomp 2008. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0951832010001602

[96] S. Lee, W. Choi, J. H. J., and D. H. Lee, "T-Box: A Forensics-Enabled Trusted Automotive Data Recording Method," *IEEE Access*, vol. 7, pp. 49 738–49 755, 2019.

[97] H. Mansor, K. Markantonakis, R. N. Akram, K. Mayes, and I. Gurulian, "Log your car: The non-invasive vehicle forensics," in *2016 IEEE Trustcom/BigDataSE/ISPA*, 2016, pp. 974–982.

[98] D. K. Nilsson and U. E. Larson, "Conducting forensic investigations of cyber attacks on automobile in-vehicle networks," in *Proceedings of the 1st International Conference on Forensic Applications and Techniques in Telecommunications, Information, and Multimedia and Workshop*, ser. e-Forensics '08. Brussels, BEL: ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2008.

[99] W. Bortles, S. McDonough, C. Smith, and M. Stogsdill, "An introduction to the forensic acquisition of passenger vehicle infotainment and telematics systems data," SAE Technical Paper, Tech. Rep., 2017.

[100] X. Yuan, P. He, Q. Zhu, and X. Li, "Adversarial Examples: Attacks and Defenses for Deep Learning," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 30, no. 9, pp. 2805–2824, 2019.

[101] O. Maler and D. Nickovic, "Monitoring temporal properties of continuous signals," in *Formal Techniques, Modelling and Analysis of Timed and Fault-Tolerant Systems.* Springer, 2004, pp. 152–166.

[102] H. Debar, M. Dacier, and A. Wespi, "Towards a taxonomy of intrusion-detection systems," *Computer Networks*, vol. 31, no. 8, pp. 805 – 822, 1999. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1389128698000176

[103] G. Welch and G. Bishop, "An Introduction to the Kalman filter," 1995.

[104] R. E. Kalman, "A new approach to linear filtering and prediction problems," 1960.

[105] Burdi Motorworks, "Mercedes Limp Home Mode," https://burdimotors.com/2017/11/30/mercedes-limp-home-mode, Accessed 2020-03-16, 2018. [Online]. Available: https://burdimotors.com/2017/11/30/mercedes-limp-home-mode

[106] S. Checkoway, D. McCoy, B. Kantor, D. Anderson, H. Shacham, S. Savage, K. Koscher *et al.*, "Comprehensive experimental analyses of automotive attack surfaces." in *USENIX Security Symposium.* San Francisco, 2011.

[107] C. Miller and C. Valasek, "Remote exploitation of an unaltered passenger vehicle," *Black Hat USA*, vol. 2015, 2015.

[108] G. Macher, E. Armengaud, E. Brenner, and C. Kreiner, "Threat and risk assessment methodologies in the automotive domain," *Procedia Computer Science*, vol. 83, pp. 1288–1294, 2016.

[109] O. Henniger, L. Apvrille, A. Fuchs, Y. Roudier, A. Ruddle, and B. Weylr, "Security requirements for automotive on-board networks," in *2009 9th International Conference on Intelligent Transport Systems Telecommunications, (ITST).* Institute of Electrical and Electronics Engineers (IEEE), oct 2009.

[110] T. Rosenstatter and T. Olovsson, "Towards a Standardized Mapping from Automotive Security Levels to Security Mechanisms," in *2018 21st International Conference on Intelligent Transportation Systems (ITSC)*, Nov 2018, pp. 1501–1507.

[111] V. H. Le, J. den Hartog, and N. Zannone, "Security and privacy for innovative automotive applications: A survey," *Computer Communications*, vol. 132, pp. 17 – 41, 2018. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S014036641731174X

[112] D. Bodeau and R. Graubart, "Cyber Resiliency Engineering Framework (MITRE Technical Report MTR1-10237)," *Bedford, MA: MITRE Corporation*, 2011.

[113] B. Baudry and M. Monperrus, "The multiple facets of software diversity: Recent developments in year 2000 and beyond," *ACM Comput. Surv.*, vol. 48, no. 1, Sep. 2015. [Online]. Available: https://doi.org/10.1145/2807593

[114] N. Dragoni, S. Giallorenzo, A. L. Lafuente, M. Mazzara, F. Montesi, R. Mustafin, and L. Safina, *Microservices: Yesterday, Today, and Tomorrow.* Cham: Springer International Publishing, 2017, pp. 195–216. [Online]. Available: https://doi.org/10.1007/978-3-319-67425-4_12

[115] C. Engelmann, G. R. Vallee, T. Naughton, and S. L. Scott, "Proactive Fault Tolerance Using Preemptive Migration," in *2009 17th Euromicro International Conference on Parallel, Distributed and Network-based Processing*, 2009, pp. 252–257.

[116] R. Romagnoli, B. H. Krogh, and B. Sinopoli, "Design of Software Rejuvenation for CPS Security Using Invariant Sets," in *2019 American Control Conference (ACC)*, 2019, pp. 3740–3745.

[117] Tencent Keen Security Lab, "Experimental Security Assessment of BMW Cars: A Summary Report," https://keenlab.tencent.com/en/whitepapers/Experimental_Security_Assessment_of_BMW_Cars_by_KeenLab.pdf, 2018, accessed: 2020-09-11.

[118] S. Kamkar, "Drive it like you hacked it: New attacks and tools to wirelessly steal cars," *Presentation at DEFCON*, vol. 23, 2015.

[119] CVE Details, “Security vunerabilities bluelink,” https://www.cvedetails.com/vulnerability-list/vendor_id-16402/product_id-37376/Hyundaiusa-Blue-Link.html, accessed: 2020-09-11.

[120] CVE List, “CVE-2019-9493,” https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2019-9493, accessed: 2020-09-11.

[121] UNECE, “Draft recommendation on cyber security of the task force on cyber security and over-the-air issues of UNECE wp.29 GRVA,” UNECE, Tech. Rep., 2018.

[122] “ISO/SAE 21434 Road Vehicles – Cybersecurity Engineering,” International Organization for Standardization (ISO), Standard, 2020.

[123] “Good practices for security of smart cars,” ENISA, Tech. Rep., 2019.

[124] “Cyber security and resilience of smart cars,” ENISA, Tech. Rep., 2016.

[125] “SAE J3061: Cybersecurity guidebook for cyber-physical vehicle systems,” SAE International, Standard, 2016.

[126] EVITA, “EVITA deliverables,” https://www.evita-project.org/deliverables.html, accessed: 2020-09-11.

[127] M. Islam, A. Lautenbach, C. Sandberg, and T. Olovsson, “A risk assessment framework for automotive embedded systems,” *Proceedings of the 2nd ACM International Workshop on Cyber-Physical System Security*, 2016.

[128] Microsoft, “The STRIDE threat model,” https://msdn.microsoft.com/en-us/library/ee823878.aspx, 2005, accessed: 2020-09-11.

[129] T. Rosenstatter and T. Olovsson, “Towards a standardized mapping from automotive security levels to security mechanisms,” in *2018 21st International Conference on Intelligent Transportation Systems (ITSC)*, 2018, pp. 1501–1507.

[130] A. Karahasanovic, P. Kleberger, and M. Almgren, “Adapting threat modeling methods for the automotive industry,” *15th ESCAR, Berlin*, 2017.

[131] “ISO 26262:2011 Road Vehicles – Functional Safety,” International Organization for Standardization (ISO), Standard, 2011.

[132] F. D. Garcia, D. Oswald, T. Kasper, and P. Pavlidès, “Lock it and still lose it—on the (in) security of automotive remote keyless entry systems,” in *25th USENIX Security Symposium (USENIX Security 16)*, 2016.

[133] A. Greenberg, “Just a pair of these $11 radio gadgets can steal a car,” https://www.wired.com/2017/04/just-pair-11-radio-gadgets-can-steal-car/, accessed: 2020-09-11.

[134] A. Francillon, B. Danev, and S. Capkun, “Relay attacks on passive keyless entry and start systems in modern cars,” in *Proceedings of the Network and Distributed System Security Symposium (NDSS).* ETH Zürich, Department of Computer Science, 2011.

[135] M. L. Psiaki and T. E. Humphreys, "GNSS spoofing and detection," *Proceedings of the IEEE*, vol. 104, no. 6, pp. 1258–1270, 2016.

[136] K. Iehira, H. Inoue, and K. Ishida, "Spoofing attack using bus-off attacks against a specific ECU of the CAN bus," in *15th IEEE Consumer Communications & Networking Conference (CCNC)*, 2018, pp. 1–4.

[137] Q. Meng, L. Hsu, B. Xu, X. Luo, and A. El-Mowafy, "A GPS spoofing generator using an open sourced vector tracking-based receiver," *Sensors*, vol. 19, no. 18, p. 3993, 2019.

[138] CVE List, "CVE-2019-12797," https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2019-12797, accessed: 2020-09-11.

[139] Y. Cao, C. Xiao, B. Cyr, Y. Zhou, W. Park *et al.*, "Adversarial sensor attack on LiDAR-based perception in autonomous driving," in *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '19. New York, NY, USA: Association for Computing Machinery, 2019, p. 2267–2281.

[140] Cyware Hacker News, "Seven car manufacturers hit by GPS spoofing attacks," https://cyware.com/news/seven-car-manufacturers-hit-by-gps-spoofing-attacks-146701c4, accessed: 2020-09-11.

[141] Help Net Security, "Research shows Tesla Model 3 and Model S are vulnerable to GPS spoofing attacks," https://www.helpnetsecurity.com/2019/06/19/tesla-gps-spoofing-attacks/, accessed: 2020-09-11.

[142] CVE, "CVE-2018-11478," https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2018-11478, accessed: 2020-09-11.

[143] D. Schmidt, K. Radke, S. Camtepe, E. Foo, and M. Ren, "A survey and analysis of the GNSS spoofing threat and countermeasures," *ACM Comput. Surv.*, vol. 48, no. 4, May 2016. [Online]. Available: https://doi.org/10.1145/2897166

[144] Pen Test Partners, "Hacking the Mitsubishi Outlander PHEV hybrid," https://www.pentestpartners.com/security-blog/hacking-the-mitsubishi-outlander-phev-hybrid-suv/, accessed: 2020-09-11.

[145] J. Petit, B. Stottelaar, M. Feiri, and F. Kargl, "Remote attacks on automated vehicles sensors: Experiments on camera and lidar," *Black Hat Europe*, vol. 11, p. 2015, 2015.

[146] C. Yan, W. Xu, and J. Liu, "Can you trust autonomous vehicles: Contactless attacks against sensors of self-driving vehicle," *DEF CON*, vol. 24, no. 8, p. 109, 2016.

[147] Tencent Keen Security Lab, "Experimental security assessment on lexus cars," https://keenlab.tencent.com/en/2020/03/30/Tencent-Keen-Security-Lab-Experimental-Security-Assessment-on-Lexus-Cars/, 2020, accessed: 2020-09-15.

[148] P. Murvay and B. Groza, "Practical security exploits of the FlexRay in-vehicle communication protocol," *International Conference on Risks and Security of Internet and Systems*, pp. 172–187, 2019.

[149] Argus Cyber Security, "A remote attack on the Bosch Drivelog connector dongle," https://argus-sec.com/remote-attack-bosch-drivelog-connector-dongle/, accessed: 2020-09-11.

[150] CVE List, "CVE-2016-9337," https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2016-9337, accessed: 2020-09-11.

[151] S. Woo, H. J. Jo, and D. H. Lee, "A practical wireless attack on the connected car and security protocol for in-vehicle can," *IEEE Transactions on Intelligent Transportation Systems*, vol. 16, no. 2, pp. 993–1006, 2015.

[152] M. Yan, J. Li, and G. Harpak, "Security Research on Mercedes-Benz: From Hardware to Car Control," https://i.blackhat.com/USA-20/Thursday/us-20-Yan-Security-Research-On-Mercedes-Benz-From-Hardware-To-Car-Control.pdf, 2020, accessed: 2020-09-15.

[153] G. H. Ruffo, "Tesla Data Leak: Old Components With Personal Info Find Their Way On eBay," https://insideevs.com/news/419525/tesla-data-leak-personal-info-ebay/, accessed: 2020-09-11.

[154] CVE List, "CVE-2018-11477," https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2018-11477, accessed: 2020-09-11.

[155] J. Liu, S. Zhang, W. Sun, and Y. Shi, "In-vehicle network attacks and countermeasures challenges and future directions," *IEEE Network*, 2017.

[156] J. C. Norte, "Hacking industrial vehicles from the internet," http://jcarlosnorte.com/security/2016/03/06/hacking-tachographs-from-the-internets.html, accessed: 2020-09-11.

[157] A. Palanca1, E. Evenchick, F. Maggi, and S. Zanero, "A stealth, selective, link-layer denial-of-service attack against automotive networks," *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*, 2017.

[158] P. Murvay and B. Groza, "DoS attacks on controller area networks by fault injections from the software layer," *Proceedings of the 12th International Conference on Availability, Reliability and Security*, 2017.

[159] CVE List, "CVE-2016-2354," https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2016-2354, accessed: 2020-09-11.

[160] K. Cho and K. Shin, "Error handling of in-vehicle networks makes them vulnerable," *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, 2016.

[161] J. Dürrwang, J. Braun, M. Rumez, and R. Kriesten, "Security evaluation of an airbag-ECU by reusing threat modeling artefacts," in *2017 International Conference on Computational Science and Computational Intelligence (CSCI)*, 2017, pp. 37–43.

[162] T. Brewster, "BMW updates kills bug in 2.2 million cars that left doors wide open to hackers," https://www.forbes.com/sites/thomasbrewster/2015/02/02/bmw-door-hacking/, 2015, accessed: 2020-09-11.

[163] T. Hunt, "Controlling vehicle features of Nissan LEAFs across the globe via vulnerable APIs," https://www.troyhunt.com/controlling-vehicle-features-of-nissan/, 2016, accessed: 2020-09-11.

[164] S. Sanwald, L. Kaneti, M. Stöttinger, and M. Böhner, "Secure boot revisited," *17th escar Europe*, 2019.

[165] *MISRA C: Guidelines for the Use of the C Language in Critical Systems 2012.* Motor Industry Research Association, 2013.

[166] T. Karthik, A. Brown, S. Awwad, D. McCoy, R. Bielawski *et al.*, "Uptane: Securing software updates for automobiles," *14th ESCAR Europe*, 2016.

[167] H. Debar, M. Dacier, and A. Wespi, "Towards a taxonomy of intrusion-detection systems," *Computer Networks*, vol. 31, no. 8, pp. 805 – 822, 1999.

[168] C. G. Rieger, D. I. Gertman, and M. A. McQueen, "Resilient control systems: Next generation design research," in *2009 2nd Conference on Human System Interactions*, 2009, pp. 632–636.

[169] R. Pallierer and B. Schmelz, "Combine AUTOSAR Standards for High-Performance In-Car Computers," https://innovation-destination.com/2017/12/13/combine-autosar-standards-high-performance-car-computers/, 2017, accessed: 2021-09-20.

[170] M. A. Rahim, M. A. Rahman, M. Rahman, A. T. Asyhari, M. Z. A. Bhuiyan, and D. Ramasamy, "Evolution of iot-enabled connectivity and applications in automotive industry: A review," *Vehicular Communications*, vol. 27, p. 100285, 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2214209620300565

[171] S. Sharma and B. Kaushik, "A survey on internet of vehicles: Applications, security issues & solutions," *Vehicular Communications*, vol. 20, 2019.

[172] M. Stoyanova, Y. Nikoloudakis, S. Panagiotakis, E. Pallis, and E. K. Markakis, "A survey on the internet of things (iot) forensics: Challenges, approaches, and open issues," *IEEE Communications Surveys Tutorials*, vol. 22, no. 2, pp. 1191–1221, 2020.

[173] Official Journal of the European Union, "DIRECTIVE 2010/40/EU OF THE EUROPEAN PARLIAMENT AND OF THE COUNCIL," https://eur-lex.europa.eu/LexUriServ/LexUriServ.do?uri=OJ:L:2010:207:0001:0013:EN:PDF/, 2010, accessed: 2022-02-02.

[174] B. Nelsons, A. Philips, and C. Steuart, *Guide to computer forensics and investigations.* Cengage, 2018.

[175] A. MacDermott, T. Baker, P. Buck, F. Iqbal, and Q. Shi, "The internet of things: Challenges and considerations for cybercrime investigations and digital forensics," *International Journal of Digital Crime and Forensics*, vol. 12, pp. 1–13, 06 2019.

[176] ENISA, "Is Software More Vulnerable Today?" https://www.enisa.europa.eu/publications/info-notes/is-software-more-vulnerable-today, 2018, accessed: 2021-02-03.

[177] Khanapuri, Eshaan, Chintalapati, Veera Venkata Tarun Kartik, Sharma, Rajnikant, and Gerdes, Ryan, "Learning based longitudinal vehicle platooning threat detection, identification and mitigation," *IEEE Transactions on Intelligent Vehicles*, 2021, doi=10.1109/TIV.2021.3122144.

[178] K. Strandberg, T. Rosenstatter, Rodi Jolak, Nasser Nowdehi, and Tomas Olovsson, "Resilient Shield: Reinforcing the Resilience of Vehicles Against Security Threats," *IEEE 93rd Vehicle Technology Conference*, 2021.

[179] X. Feng, E. S. Dawam, and D. Li, "Autonomous vehicles' forensics in smart cities," in *2019 IEEE SmartWorld, Ubiquitous Intelligence Computing, Advanced Trusted Computing, Scalable Computing Communications, Cloud Big Data Computing, Internet of People and Smart City Innovation (SmartWorld/SCALCOM/UIC/ATC/CBDCom/IOP/SCI)*, 2019, pp. 1688–1694.

[180] N. Vinzenz and T. Eggendorfer, "Proposal for a secure forensic data storage," 2020.

[181] C. Huang, R. Lu, and K. R. Choo, "Vehicular fog computing: Architecture, use case, and security and forensic challenges," *IEEE Communications Magazine*, vol. 55, no. 11, pp. 105–111, 2017.

[182] Electronic Privacy Information Center, "The Drivers Privacy Protection Act (DPPA) and the Privacy of Your State Motor Vehicle Record," https://epic.org/privacy/drivers/, 2020, accessed: 2020-11-17.

[183] ——, "Automobile Event Data Recorders (Black Boxes) and Privacy," https://epic.org/privacy/edrs/, 2020, accessed: 2020-11-17.

[184] M. Cebe, E. Erdin, K. Akkaya, H. Aksu, and S. Uluagac, "Block4forensic: An integrated lightweight blockchain framework for forensics applications of connected vehicles," *IEEE Communications Magazine*, vol. 56, no. 10, 2018.

[185] H. Mansor, K. Markantonakisy, R. Akramz, K. Mayesx, and I. Gurulian, "Log your car: The non-invasive vehicle forensics," *IEEE International Conference on Trust, Security and Privacy in Computing and Communications*, 2016.

[186] K. K. Gomez Buquerin, C. Corbett, and H.-J. Hof, "A generalized approach to automotive forensics," *Forensic Science International: Digital Investigation*, vol. 36, p. 301111, 2021, dFRWS 2021 EU - Selected Papers and Extended Abstracts of the Eighth

Annual DFRWS Europe Conference. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2666281721000056

[187] National Institute of Standards and Technology, "Guidelines on Mobile Device Forensics," https://csrc.nist.gov/publications/detail/sp/800-72/final, 2014, accessed: 2021-02-02.

[188] ——, "Guide to Integrating Forensic Techniques into Incident Response," https://csrc.nist.gov/publications/detail/sp/800-86/final, 2006, accessed: 2020-10-06.

[189] International Organization for Standardization, "Information technology — Security techniques — Guidelines for identification, collection, acquisition and preservation of digital evidence," https://www.iso.org/standard/44381.html, 2018, accessed: 2020-10-06.

[190] National Institute of Standards and Technology, "Guidelines on Mobile Device Forensics," https://www.nist.gov/publications/guidelines-mobile-device-forensics, 2014, accessed: 2021-02-01.

[191] D. Jacobs, K. R. Choo, M. Kechadi, and N. Le-Khac, "Volkswagen car entertainment system forensics," in *2017 IEEE Trustcom/BigDataSE/ICESS*, 2017, pp. 699–705.

[192] M. Hossain, R. Hasan, and S. Zawoad, "Trust-iov: A trustworthy forensic investigation framework for the internet of vehicles (iov)," in *2017 IEEE International Congress on Internet of Things (ICIOT)*, 2017, pp. 25–32.

[193] X. Wang, Y. Zhou, X. Ma, N. Lu, N. Cheng, and K. Zhang, "Smart cyber forensics of rear-end collision based on multi-access edge computing," in *2019 IEEE/CIC International Conference on Communications in China (ICCC)*, 2019, pp. 1012–1017.

[194] C. Wohlin, "Guidelines for snowballing in systematic literature studies and a replication in software engineering," in *Proceedings of the 18th international conference on evaluation and assessment in software engineering*, 2014, pp. 1–10.

[195] C. Oham, R. A. Michelin, R. Jurdak, S. S. Kanhere, and S. Jha, "B-ferl: Blockchain based framework for securing smart vehicles," *Information Processing & Management*, vol. 58, no. 1, p. 102426, 2021.

[196] C. Yoon, J. Hwang, M. Cho, and B. G. Lee, "Study on did application methods for blockchain-based traffic forensic data," *Applied Sciences*, vol. 11, no. 3, 2021.

[197] C. Alexakos, C. Katsini, K. Votis, A. Lalas, D. Tzovaras, and D. Serpanos, "Enabling digital forensics readiness for internet of vehicles," *Transportation Research Procedia*, vol. 52, pp. 339–346, 2021, 23rd EURO Working Group on Transportation Meeting, EWGT 2020, 16-18 September 2020, Paphos, Cyprus.

[198] M. Waltereit, M. Uphoff, P. Zdankin, V. Matkovic, and T. Weis, “A digital forensic approach for optimizing the investigation of hit-and-run accidents,” in *Digital Forensics and Cyber Crime*, S. Goel, P. Gladyshev, D. Johnson, M. Pourzandi, and S. Majumdar, Eds. Springer International Publishing, 2021, pp. 204–223.

[199] P. A. Abhay, N. V. Jishnu, K. T. Meenakshi, P. S. Yaswanth, and A. O. Philip, “Auto block IoT: A forensics framework for connected vehicles,” *Journal of Physics: Conference Series*, vol. 1911, no. 1, p. 012002, may 2021. [Online]. Available: https://doi.org/10.1088/1742-6596/1911/1/012002

[200] M. A. Hoque and R. Hasan, “Avguard: A forensic investigation framework for autonomous vehicles,” 02 2021.

[201] J. Daily, M. DiSogra, and D. Van, “Chip and board level digital forensics of cummins heavy vehicle event data recorders,” *SAE Int. J. Adv. & Curr. Prac. in Mobility*, vol. 2, pp. 2374–2388, 04 2020.

[202] P. Sharma, U. Siddanagaiah, and G. Kul, “Towards an ai-based after-collision forensic analysis protocol for autonomous vehicles,” in *2020 IEEE Security and Privacy Workshops (SPW)*, 2020, pp. 240–243.

[203] H. Guo, W. Li, M. Nejad, and C. C. Shen, “Proof-of-event recording system for autonomous vehicles: A blockchain-based solution,” *IEEE Access*, vol. 8, pp. 182 776–182 786, 2020.

[204] A. Philip and R. Saravanaguru, “Secure incident & evidence management framework (siemf) for internet of vehicles using deep learning and blockchain,” *Open Computer Science*, vol. 10, p. 408, 11 2020.

[205] M. Li, J. Weng, J.-N. Liu, X. Lin, and C. Obimbo, “Towards vehicular digital forensics from decentralized trust: An accountable, privacy-preservation, and secure realization,” 2020.

[206] Z. Ma, M. Jiang, and W. Huang, “Trusted forensics scheme based on digital watermark algorithm in intelligent vanet,” *Neural Computing and Applications*, vol. 32, 03 2020.

[207] A. Mehrish, P. Singh, P. Jain, A. V. Subramanyam, and M. Kankanhalli, “Egocentric analysis of dash-cam videos for vehicle forensics,” *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 30, no. 9, pp. 3000–3014, 2020.

[208] M. Waltereit and T. Weis, “An approach to exonerate innocent suspects in hit-and-run accidents via route reconstruction,” in *2019 IEEE International Conference on Pervasive Computing and Communications Workshops (PerCom Workshops)*, 2019, pp. 447–448.

[209] K. Bahirat, N. Vaishnav, S. Sukumaran, and B. Prabhakaran, “Add-far: Attacked driving dataset for forensics analysis and research,” in *Proceedings of the 10th ACM Multimedia Systems Conference*, ser. MMSys ’19. New York, NY, USA: Association for Computing Machinery, 2019, p. 243–248. [Online]. Available: https://doi.org/10.1145/3304109.3325817

[210] L. Cintron, S. Graham, D. Hodson, and B. Mullins, "Modeling liability data collection systems for intelligent transportation infrastructure using hyperledger fabric," in *Critical Infrastructure Protection XIII*, J. Staggs and S. Shenoi, Eds. Springer International Publishing, 2019, pp. 137–156.

[211] S. LEE, W. CHO, H. J. JO, and D. H. LEE, "T-box: A forensics-enabled trusted automotive data recording method," *IEEE Access*, vol. 7, pp. 49 738–49 755, 2019.

[212] L. Davi, D. Hatebur, M. Heisel, and R. Wirtz, "Combining safety and security in autonomous cars using blockchain technologies," in *Computer Safety, Reliability, and Security*, A. Romanovsky, E. Troubitsyna, I. Gashi, E. Schoitsch, and F. Bitsch, Eds. Cham: Springer International Publishing, 2019, pp. 223–234.

[213] D. Billard and B. Bartolomei, "Digital forensics and privacy-by-design: Example in a blockchain-based dynamic navigation system," in *Privacy Technologies and Policy*, M. Naldi, G. F. Italiano, K. Rannenberg, M. Medina, and A. Bourka, Eds. Cham: Springer International Publishing, 2019, pp. 151–160.

[214] M. Ugwu, C. Oham, I. Nwakanma, and O. Izunna, "A tiered blockchain framework for vehicular forensics," *International Journal of Network Security & Its Applications*, vol. 10, pp. 25–34, 09 2018.

[215] M. Marchetti and D. Stabili, "Read: Reverse engineering of automotive data frames," *IEEE Transactions on Information Forensics and Security*, vol. 14, no. 4, pp. 1083–1097, 2019.

[216] H. Guo, E. Meamari, and C. Shen, "Blockchain-inspired event recording system for autonomous vehicles," in *2018 1st IEEE International Conference on Hot Information-Centric Networking (HotICN)*, 2018, pp. 218–222.

[217] R. Hussain, D. Kim, J. Son, J. Lee, C. A. Kerrache, A. Benslimane, and H. Oh, "Secure and privacy-aware incentives-based witness service in social internet of vehicles clouds," *IEEE Internet of Things Journal*, vol. 5, no. 4, pp. 2441–2448, 2018.

[218] A. Mehrish, A. V. Subramanyam, and M. Kankanhalli, "Multimedia signatures for vehicle forensics," in *2017 IEEE International Conference on Multimedia and Expo (ICME)*, 2017, pp. 685–690.

[219] X. Feng, E. S. Dawam, and S. Amin, "A new digital forensics model of smart city automated vehicles," in *2017 IEEE International Conference on Internet of Things (iThings)*, 2017, pp. 274–279.

[220] A. D. Sathe and V. D. Deshmukh, "Advance vehicle-road interaction and vehicle monitoring system using smart phone applications," in *2016 Online International Conference on Green Engineering and Technologies (IC-GET)*, 2016, pp. 1–6.

[221] N. Watthanawisuth, T. Lomas, and A. Tuantranont, “Wireless black box using mems accelerometer and gps tracking for accidental monitoring of vehicles,” in *Proceedings of 2012 IEEE-EMBS International Conference on Biomedical and Health Informatics*, 2012, pp. 847–850.

[222] D. K. Nilsson and U. E. Larson, “Conducting forensic investigations of cyber attacks on automobile in-vehicle networks,” ser. e-Forensics ’08. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2008.

[223] A. Attenberger, “Data sources for information extraction in automotive forensics,” in *Computer Aided Systems Theory – EUROCAST 2019*, R. Moreno-Díaz, F. Pichler, and A. Quesada-Arencibia, Eds. Cham: Springer International Publishing, 2020, pp. 137–144.

[224] R. Rak and D. Kopencova, “Actual issues of modern digital vehicle forensic,” *Internet of Things and Cloud Computing*, vol. 8, p. 12, 01 2020.

[225] D. Kopencova and R. Rak, “Issues of vehicle digital forensics,” in *2020 XII International Science-Technical Conference AUTOMOTIVE SAFETY*, 2020, pp. 1–6.

[226] H. S. Lallie, “Dashcam forensics: A preliminary analysis of 7 dashcam devices,” *Forensic Science International: Digital Investigation*, vol. 33, p. 200910, 2020.

[227] K. Dološ, C. Meyer, A. Attenberger, and J. Steinberger, “Driver identification using in-vehicle digital data in the forensic context of a hit and run accident,” *Forensic Science International: Digital Investigation*, vol. 35, p. 301090, 2020.

[228] N.-A. Le-Khac, D. Jacobs, J. Nijhoff, K. Bertens, and K.-K. R. Choo, “Smart vehicle forensics: Challenges and case study,” *Future Generation Computer Systems*, vol. 109, pp. 500–510, 2020.

[229] D. Steiner, L. Chen, D. Hayes, and N.-A. Le-Khac, “Vehicle communication within networks -investigation and analysis approach: A case study,” *Annual ADFSL Conference on Digital Forensics, Security and Law*, 05 2019.

[230] D. Sladović, D. Topolčić, K. Hausknecht, and G. Sirovatka, “Investigating modern cars,” in *2019 42nd International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*, 2019, pp. 1159–1164.

[231] N. Vinzenz and T. Eggendorfer, “Forensic investigations in vehicle data stores,” ser. CECC 2019. New York, NY, USA: Association for Computing Machinery, 2019. [Online]. Available: https://doi.org/10.1145/3360664.3360665

[232] M. Hussain, M. Beg, M. Alam, and S. Laskar, “Big data analytics platforms for electric vehicle integration in transport oriented smart cities: Computing platforms for platforms for electric vehicle integration

in smart cities," *International Journal of Digital Crime and Forensics*, vol. 11, pp. 23–42, 07 2019.

[233] C. Urquhart, X. Bellekens, C. Tachtatzis, R. Atkinson, H. Hindy, and A. Seeam, "Cyber-security internals of a skoda octavia vrs: A hands on approach," *IEEE Access*, vol. 7, pp. 146 057–146 069, 2019.

[234] C. J. Whelan, J. Sammons, B. McManus, and T. Fenger, "Retrieval of infotainment system artifacts from vehicles using ive," 2018.

[235] A. Koch, R. Altschaffel, S. Kiltz, M. Hildebrandt, and J. Dittmann, "Exploring the processing of personal data in modern vehicles - a proposal of a testbed for explorative research to achieve transparency for privacy and security," in *2018 11th International Conference on IT Security Incident Management IT Forensics (IMF)*, 2018, pp. 15–26.

[236] S. Tatjana, B. Ištvan, T. Nena, and K. Biljana, "Application of digital forensics in traffic conditions," in *2018 23rd International Scientific-Professional Conference on Information Technology (IT)*, 2018, pp. 1–4.

[237] F. Leuzzi, E. Del Signore, and R. Ferranti, "Towards a pervasive and predictive traffic police," in *Traffic Mining Applied to Police Activities*, F. Leuzzi and S. Ferilli, Eds. Cham: Springer International Publishing, 2018, pp. 19–35.

[238] C. Ivan, P. Dragan, P. Marko, and H. Sinisa, "Application possibilities of digital forensic procedures in vehicle telematics systems," vol. 10, pp. 133–144, 2018. [Online]. Available: https://yadda.icm.edu.pl/baztech/element/bwmeta1.element.baztech-99e384a5-27b4-4c1a-9c61-f9b63a138a27/c/Art._10.pdf

[239] Z. A. Baig, P. Szewczyk, C. Valli, P. Rabadia, P. Hannay, M. Chernyshev, M. Johnstone, P. Kerai, A. Ibrahim, K. Sansurooah, N. Syed, and M. Peacock, "Future challenges for smart cities: Cyber-security and digital forensics," *Digital Investigation*, vol. 22, pp. 3–13, 2017. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1742287617300579

[240] R. Altschaffel, K. Lamshöft, S. Kiltz, and J. Dittmann, "A survey on open automotive forensics." SECURWARE 2017 The Eleventh International Conference on Emerging Security Information, 04 2017.

[241] W. Bortles, S. McDonough, C. Smith, and M. Stogsdill, "An introduction to the forensic acquisition of passenger vehicle infotainment and telematics systems data," 2017.

[242] J. Lacroix, K. El-Khatib, and R. Akalu, "Vehicular digital forensics: What does my vehicle know about me?" in *Proceedings of the 6th ACM Symposium on Development and Analysis of Intelligent Vehicular Networks and Applications*, ser. DIVANet '16. New York, NY, USA: Association for Computing Machinery, 2016, p. 59–66. [Online]. Available: https://doi.org/10.1145/2989275.2989282

[243] J. S. Ogden and M. Martonovich, “Forensic engineering tools and analysis of heavy vehicle event data recorders (hvedrs),” *Journal of the National Academy of Forensic Engineers*, vol. 33, no. 2, Jan. 2016.

[244] N. Krishnamurthy and J. H. Hansen, “Car noise verification and applications,” *Int. J. Speech Technol.*, vol. 17, no. 2, p. 167–181, Jun. 2014.

[245] D.-W. Park, “Forensic analysis technique of car black box,” vol. 8, pp. 1–10, 01 2014.

[246] K.-S. Lim, C. Lee, J. Park, and S. Lee, “Test-driven forensic analysis of satellite automotive navigation systems,” *Journal of Intelligent Manufacturing*, vol. 25, 04 2014.

[247] J. Johnson, J. Daily, and A. Kongs, “On the digital forensics of heavy truck electronic control modules,” *SAE International Journal of Commercial Vehicles*, vol. 7, no. 1, pp. 72–88, apr 2014. [Online]. Available: https://doi.org/10.4271/2014-01-0495

[248] T. Hoppe, S. Kuhlmann, S. Kiltz, and J. Dittmann, “It-forensic automotive investigations on the example of route reconstruction on automotive system and communication data,” vol. 7612, 09 2012, pp. 125–136.

[249] S. Al-Kuwari and S. D. Wolthusen, “On the feasibility of carrying out live real-time forensics for modern intelligent vehicles,” in *Forensics in Telecommunications, Information, and Multimedia*, X. Lai, D. Gu, B. Jin, Y. Wang, and H. Li, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 207–223.

[250] D. K. Nilsson and U. E. Larson, “Conducting forensic investigations of cyber attacks on automobile in-vehicle networks,” in *Proceedings of the 1st International Conference on Forensic Applications and Techniques in Telecommunications, Information, and Multimedia and Workshop*, ser. e-Forensics ’08. Brussels, BEL: ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2008.

[251] J. S. Daily, N. Singleton, B. Downing, and G. W. Manes*, “Light vehicle event data recorder forensics,” in *Advances in Computer and Information Sciences and Engineering*, T. Sobh, Ed. Dordrecht: Springer Netherlands, 2008, pp. 172–177.

[252] D. K. Nilsson and U. E. Larson, “Combining physical and digital evidence in vehicle environments,” in *2008 Third International Workshop on Systematic Approaches to Digital Forensic Engineering*, 2008, pp. 10–14.

[253] Alientech, “Tools for ECU remapping,” https://www.alientech-tools.com/, 2022, accessed: 2022-02-09.

[254] P&E Microcomputer Systems Inc., “Easily Manage i.MX RT Secure Boot for Production Programming,” https://www.pemicro.com/, 2022, accessed: 2022-02-09.

[255] Berla Corporation, "iVe 3.5," https://berla.co/, 2022, accessed: 2022-02-09.

[256] Volvo Car Corporation, "Volvo On Call," visited on 2021-07-03. [Online]. Available: https://www.volvocars.com/intl/v/volvo-cars-app

[257] OnStar Corporation, "Welcome to onStar," visited on 2021-07-03. [Online]. Available: https://www.onstar.com/us/en/home/

[258] Hacking and Countermeasure Research Lab, "DRIVING DATASET," https://ocslab.hksecurity.net/Datasets/driving-dataset, 2018, accessed: 2021-05-18.

[259] CASE, "An international standard supporting automated combination, validation, and analysis of cyber-investigation information," https://caseontology.org/, 2022, accessed: 2022-02-07.

[260] B. H. Bloom, "Space/time trade-offs in hash coding with allowable errors," *Commun. ACM*, vol. 13, no. 7, p. 422–426, Jul. 1970. [Online]. Available: https://doi.org/10.1145/362686.362692

[261] A. Valjarevic and H. Venter, "A harmonized process model for digital forensic investigation readiness," in *Advances in Digital Forensics IX*, G. Peterson and S. Shenoi, Eds. Springer Berlin Heidelberg, 2013, pp. 67–82.

[262] CORDIS, "A Novel Adaptive Cybersecurity Framework for the Internet-of-Vehicles," https://cordis.europa.eu/project/id/833742, 2006, accessed: 2021-04-21.

[263] S. Kiltz, J. Dittmann, and C. Vielhauer, "Supporting forensic design - a course profile to teach forensics," 05 2015, pp. 85–95.

[264] K. Chae, D. Kim, S. Jung, J. Choi, and S. Jung, "Evidence collecting system from car black boxes," in *2010 7th IEEE Consumer Communications and Networking Conference*, 2010, pp. 1–2.

[265] B. Canis and D. R. Peterman, ""Black Boxes" in Passenger Vehicles:Policy Issues," https://fas.org/sgp/crs/misc/R43651.pdf, 2014, accessed: 2020-11-17.

[266] C. Patsakis and A. Solanas, "Privacy-aware event data recorders: cryptography meets the automotive industry again," *IEEE Communications Magazine*, vol. 51, no. 12, pp. 122–128, 2013.

[267] Volvo Car Corporation, "Volvo Cars and Uber present first autonomous drive-ready production car," https://group.volvocars.com/news/future-mobility/2019/volvo-and-uber-present-autonomous-drive-ready-xc90, 2019, accessed: 2021-06-02.

[268] Volvo Cars, "Volvo Cars teams up with world's leading mobility technology platform DiDi for self-driving test fleet," https://www.media.volvocars.com/global/en-gb/media/pressreleases/280668/volvo-cars-teams-up-with-worlds-leading-mobility-technology-platform-didi-for-self-driving-test-flee, 2021, accessed: 2021-06-07.

[269] yubico, “Protect your digital world with YubiKey,” https://www.yubico.com/, 2021, accessed: 2021-06-02.

[270] J. Samuel, N. Mathewson, J. Cappos, and R. Dingledine, “Survivable key compromise in software update systems,” 12 2010, pp. 61–72.

[271] 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,” in *Communication Technologies for Vehicles*, T. Strang, A. Festag, A. Vinel, R. Mehmood, C. Rico Garcia, and M. Röckl, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 224–238.

[272] S. Mahmud, S. Shanker, and I. Hossain, “Secure software upload in an intelligent vehicle via wireless communication links,” in *IEEE Proceedings. Intelligent Vehicles Symposium, 2005.*, 2005, pp. 588–593.

[273] M. Steger, C. A. Boano, T. Niedermayr, M. Karner, J. Hillebrand, K. Roemer, and W. Rom, “An efficient and secure automotive wireless software update framework,” *IEEE Transactions on Industrial Informatics*, vol. 14, no. 5, pp. 2181–2193, 2018.

[274] D. K. Nilsson and U. E. Larson, “Secure firmware updates over the air in intelligent vehicles,” in *ICC Workshops - 2008 IEEE International Conference on Communications Workshops*, 2008, pp. 380–384.

[275] D. K. Nilsson, L. Sun, and T. Nakajima, “A framework for self-verification of firmware updates over the air in vehicle ecus,” in *2008 IEEE Globecom Workshops*, 2008, pp. 1–5.

[276] K. Strandberg, T. Rosenstatter, R. Jolak, N. Nowdehi, and T. Olovsson, “Resilient shield: Reinforcing the resilience of vehicles against security threats,” 04 2021, pp. 1–7.

[277] T. Kelly and R. Weaver, “The goal structuring notation–a safety argument notation,” *Proc Dependable Syst Networks Workshop Assurance Cases*, 01 2004.

[278] S. Mahmud, S. Shanker, and I. Hossain, “Secure software upload in an intelligent vehicle via wireless communication links,” in *IEEE Proceedings. Intelligent Vehicles Symposium, 2005.*, 2005, pp. 588–593.

[279] 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,” 03 2011, pp. 224–238.

[280] D. K. Nilsson and U. E. Larson, “Secure firmware updates over the air in intelligent vehicles,” in *ICC Workshops - 2008 IEEE International Conference on Communications Workshops*, 2008, pp. 380–384.

[281] J. Samuel, N. Mathewson, J. Cappos, and R. Dingledine, “Survivable key compromise in software update systems,” 12 2010, pp. 61–72.

[282] T. Karthik, Kuppusamy, and D. McCoy, "Uptane : Securing software updates for automobiles," 2016.

[283] Association of Chief Police Officers, "Acpo good practice guide for digital evidence," 2012.

[284] Experience Per Mile Advisory Council, "Share of new vehicles shipped worldwide with built-in connectivity in 2020 and 2030," May 2020. [Online]. Available: https://www.statista.com/statistics/1276018/share-of-connected-cars-in-total-new-car-sales-worldwide/

[285] F. Narisawa, Y. Asada, T. Sobue, M. Yano, O. Sakanoue, K. Maeda, and M. Saito, "Vehicle Electronic Control Units for Autonomous Driving in Safety and Comfort," *Hitachi Review*, vol. 71, no. 1, pp. 78–83, 2022.

[286] Scotiabank, "Number of cars sold worldwide from 2010 to 2023, with a 2024 forecast," Feb. 2024. [Online]. Available: https://www.statista.com/statistics/200002/international-car-sales-since-1990/

[287] B. Blanchet, "Modeling and Verifying Security Protocols with the Applied Pi Calculus and ProVerif," *Foundations and Trends® in Privacy and Security*, vol. 1, no. 1-2, pp. 1–135, 2016.

[288] B. Blanchet, B. Smyth, V. Cheval, and M. Sylvestre, "ProVerif 2.05: Automatic Cryptographic Protocol Verifier, User Manual and Tutoria." [Online]. Available: http://prosecco.gforge.inria.fr/personal/bblanche/proverif/manual.pdf

[289] A. Chlipala, *Certified Programming with Dependent Types: A Pragmatic Introduction to the Coq Proof Assistant.* The MIT Press, 2013.

[290] M. Wenzel, L. C. Paulson, and T. Nipkow, "The Isabelle Framework," in *Theorem Proving in Higher Order Logics*, O. A. Mohamed, C. Muñoz, and S. Tahar, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 33–38.

[291] G. Holzmann, "The model checker SPIN," *IEEE Transactions on Software Engineering*, vol. 23, no. 5, pp. 279–295, 1997.

[292] G. Behrmann, A. David, K. G. Larsen, J. Håkansson, P. Pettersson, W. Yi, and M. Hendriks, "Uppaal 4.0," 2006.

[293] L. Lamport, *Specifying Systems: The TLA+ Language and Tools for Hardware and Software Engineers.* Addison-Wesley, 6 2002. [Online]. Available: https://www.microsoft.com/en-us/research/publication/specifying-systems-the-tla-language-and-tools-for-hardware-and-software-engineers/

[294] B. Blanchet, "CryptoVerif: Computationally sound mechanized prover for cryptographic protocols," in *Dagstuhl seminar "Formal Protocol Verification Applied*, vol. 117, 2007, p. 156.

[295] S. Meier, B. Schmidt, C. Cremers, and D. Basin, "The TAMARIN Prover for the Symbolic Analysis of Security Protocols," in *Computer Aided Verification*, N. Sharygina and H. Veith, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 696–701.

[296] D. Dolev and A. Yao, "On the security of public key protocols," *IEEE Transactions on Information Theory*, vol. 29, no. 2, pp. 198–208, 1983.

[297] J. Wang, "Remote Offline Attestation for Seldomly Connected Vehicular Systems," Jan. 2024, unpublished.

[298] D. Basin, J. Dreier, L. Hirschi, S. Radomirovic, R. Sasse, and V. Stettler, "A Formal Analysis of 5G Authentication," in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '18. ACM, Oct. 2018. [Online]. Available: http://dx.doi.org/10.1145/3243734.3243846

[299] C. Cremers, B. Kiesl, and N. Medinger, "A Formal Analysis of IEEE 802.11's WPA2: Countering the Kracks Caused by Cracking the Counters," in *29th USENIX Security Symposium (USENIX Security 20)*. USENIX Association, Aug. 2020, pp. 1–17. [Online]. Available: https://www.usenix.org/conference/usenixsecurity20/presentation/cremers

[300] C. Cremers, M. Horvat, J. Hoyland, S. Scott, and T. van der Merwe, "A Comprehensive Symbolic Analysis of TLS 1.3, year = 2017," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '17. New York, NY, USA: Association for Computing Machinery, p. 1773–1788. [Online]. Available: https://doi.org/10.1145/3133956.3134063

[301] K. Bhargavan, V. Cheval, and C. Wood, "A Symbolic Analysis of Privacy for TLS 1.3 with Encrypted Client Hello," in *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '22. New York, NY, USA: Association for Computing Machinery, 2022, p. 365–379. [Online]. Available: https://doi.org/10.1145/3548606.3559360

[302] C. Ponsard and D. Darquennes, "Towards Formal Security Verification of Over-the-Air Update Protocol: Requirements, Survey and UpKit Case Study." in *Proceedings of the 7th International Conference on Information Systems Security and Privacy - Volume 1: ForSE,*, INSTICC. SciTePress, 2021, pp. 800–808.

[303] A. Langiu, C. A. Boano, M. Schuß, and K. Römer, "UpKit: An Open-Source, Portable, and Lightweight Update Framework for Constrained IoT Devices," in *2019 IEEE 39th International Conference on Distributed Computing Systems (ICDCS)*, 2019, pp. 2101–2112.

[304] D. McGrew and K. Igoe, "AES-GCM Authenticated Encryption in the Secure Real-time Transport Protocol (SRTP)," RFC 7714, Dec. 2015. [Online]. Available: https://www.rfc-editor.org/info/rfc7714

[305] C. M. Lonvick and T. Ylonen, "The Secure Shell (SSH) Transport Layer Protocol," RFC 4253, Jan. 2006. [Online]. Available: https://www.rfc-editor.org/info/rfc4253

[306] E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.3," RFC 8446, Aug. 2018. [Online]. Available: https://www.rfc-editor.org/info/rfc8446

[307] B. Campbell, J. Bradley, N. Sakimura, and T. Lodderstedt, "OAuth 2.0 Mutual-TLS Client Authentication and Certificate-Bound Access Tokens," RFC 8705, Feb. 2020. [Online]. Available: https://www.rfc-editor.org/info/rfc8705

[308] K. Strandberg, 2024, private communication.

[309] O. Gasser, R. Holz, and G. Carle, "A deeper understanding of SSH: Results from Internet-wide scans," in *2014 IEEE Network Operations and Management Symposium (NOMS)*, 2014, pp. 1–9.

[310] N. L. M. Van Adrichem, A. R. Lua, X. Wang, M. Wasif, F. Fatturrahman, and F. A. Kuipers, "DNSSEC Misconfigurations: How Incorrectly Configured Security Leads to Unreachability," in *2014 IEEE Joint Intelligence and Security Informatics Conference*, 2014, pp. 9–16.